PROYECTO DE FIN DE CARRERA

EVOLUCIÓN DE *FUNCTIONAL LINK NETWORKS* CON UN ENFOQUE MULTIOBJETIVO

Autor: Ángel Carrasco Fernández

Tutores: Ricardo Aler Mur

Inés M. Galván León

Fecha de presentación: 13/03/09

AGRADECIMIENTOS

A mis padres, a mi hermana y a todos mis familiares, que gracias a su apoyo y comprensión han permitido que pudiera dedicarme enteramente a la carrera dándome todo que necesitaba e incluso más sin pedir nada a cambio.

A mis tutores, Ricardo e Inés, que sin su inestimable ayuda y colaboración no habría sido posible la realización de este proyecto y a los que doy gracias por su paciencia y apoyo.

A mis amigos, que durante toda mi vida siempre me han apoyado y me han dado ánimos para seguir… y lo más importante, que seguirán apoyándome en mis futuros proyectos.

Y no quisiera dejar pasar la ocasión para agradecer a todos mis compañeros su colaboración a lo largo de la carrera, ya que sin ellos esto no habría sido lo mismo, ¡Gracias "Cebollitas"!

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

Existen áreas del conocimiento y el quehacer humano dentro de las cuales es común el surgimiento de problemas que consisten en la mejora de ciertas soluciones, procedimientos de los que se obtiene algún beneficio ya sea particular o común. La disciplina que se enfoca a estudiar este tipo de problemas y sus respectivas alternativas es conocida como optimización.

Al buscar la solución a un problema de optimización, existen múltiples herramientas matemáticas que nos ayudan a lograr este objetivo. No obstante, en el mundo real existen problemas de optimización para los cuales es difícil, o incluso imposible, aplicar métodos de programación matemática con éxito. En estos problemas, las heurísticas y más concretamente la computación evolutiva se vuelve una alternativa viable. Los algoritmos evolutivos surgieron y muchos investigadores los utilizan, debido principalmente a que su ventaja sobre cualquier otra técnica radica en sus resultados y a su capacidad para abstraerse del problema, es decir, que son independientes del problema a resolver. Los algoritmos evolutivos se han utilizado principalmente para resolver problemas monoobjetivo, es decir, en los que el algoritmo sólo tiene una función objetivo a minimizar.

Muchos de los problemas del mundo real involucran la optimización de varios objetivos simultáneamente (los cuales se expresan en unidades de medida diferentes o se encuentran en conflicto entre sí). La naturaleza conflictiva de sus objetivos produce un conjunto de soluciones compromiso alternativas. Uno de los enfoques actuales más exitosos para resolver eficazmente estos problemas es la utilización de los algoritmos evolutivos multiobjetivo.

El aprendizaje automático es una disciplina de la Inteligencia Artificial que permite construir modelos automáticos capaces de aprender y que se utilizan para resolver problemas de clasificación, predicción y agrupación o clustering. Son necesarios dos pasos para construir cualquier sistema de aprendizaje automático, elegir el modelo y estimar los parámetros adecuados para dicho modelo. Existen varias técnicas de aprendizaje automático que se diferencian en la forma en la que se realiza el aprendizaje por parte del modelo:

- En el aprendizaje supervisado el algoritmo produce una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema, como por ejemplo en un problema de clasificación.

- En el aprendizaje no supervisado todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema.

- En el aprendizaje por refuerzo el algoritmo aprende observando el mundo que le rodea. Su información de entrada es el feedback o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones.

La construcción de modelos puede ser tratado como un problema de optimización y por ello los algoritmos de computación evolutiva se han utilizado en el campo del

aprendizaje automático. Para ello, se deben codificar en el cromosoma los parámetros del modelo de manera que el algoritmo sea capaz de buscar el mejor modelo teniendo como objetivo minimizar el error cometido. De esta manera el método que se propone es independiente del dominio, mientras que la configuración de los parámetros del algoritmo evolutivo dependerá del dominio.

Muchos de los problemas relativos a la construcción de modelos de aprendizaje automático pueden verse desde una perspectiva multiobjetivo. Los distintos objetivos que se planteen pueden ser muy diversos, como por ejemplo en la selección de atributos, minimizando el número de atributos a seleccionar y la calidad del resultado obtenido. Los objetivos pueden estar en confrontación tratando de compensar la complejidad y la exactitud de la solución encontrada.

Hasta hace poco tiempo el enfoque multiobjetivo se abordaba intentando combinar todos los objetivos en una función de manera que el problema multiobjetivo se tratara como un problema monoobjetivo normal. Debido al amplio desarrollo de las técnicas de optimización multiobjetivo basadas en la jerarquización de Pareto, los algoritmos multiobjetivo han empezado a utilizarse dentro del campo del aprendizaje automático.

El proyecto se enmarca dentro del aprendizaje automático con una perspectiva multiobjetivo. En concreto trataremos de realizar una evolución de los functional link networks con un enfoque multiobjetivo. En trabajos anteriores [Sierra01] ya se había planteado la evolución de Functional Link Networks para determinar los atributos productos más relevantes, pero partiendo de un enfoque monoobjetivo. Este proyecto tiene el objetivo de estudiar un enfoque multiobjetivo, y para ello se ha utilizado un algoritmo genético multiobjetivo como es el NSGAII.

Como ya se ha mencionado, el método de aprendizaje automático elegido es el modelo denominado Functional Link Networks. Se caracterizan porque construyen clasificadores no lineales mediante el uso de atributos producto y el aprendizaje suele ser rápido debido a que se eliminan las capas ocultas del perceptron multicapa, de manera que el clasificador se convierte en un perceptron simple con los atributos originales sumados a los atributos producto. El hecho de que su aprendizaje sea rápido lo convierte en una técnica de clasificación interesante en el marco de la computación evolutiva, pues para cada individuo es necesario construir y entrenar el modelo. Habrá que tener en cuenta, que la función de evaluación que se aplicará a un algoritmo genético debe tener ciertas imposiciones, ya que no podemos permitir que el tiempo en realizar cada una de las evaluaciones sea enorme y por ello, se debe descartar algunas técnicas de clasificación cuyo tiempo de evaluación sea grande. De la misma manera, no podemos pretender utilizar clasificadores tan sencillos, que sus resultados no sean competitivos con otros que se pueden obtener de la literatura. Por ello, se ha decidido utilizar FLN.

En este proyecto el enfoque multiobjetivo se ha planteado de dos maneras diferentes, con la finalidad de estudiar y analizar esta perspectiva multiobjetivo. Éstas son:

- Caso 1: minimizar el porcentaje de fallos total y minimizar el número de atributos producto usados en dicha solución.

- Caso 2: minimizar el porcentaje de fallos para cada una de las clases a clasificar.

El sistema desarrollado se ha validado para varios dominios, uno sintético para mostrar que la idea que se plantea en el proyecto funciona correctamente, y tres dominios reales. Dos de ellos, el cáncer y diabetes, de la base de datos de Proben preparados por Prechelt [Prechelt94]. El tercer dominio es un problema de clasificación que se enmarca en la comunicación cerebro-ordenador (Brain Computer Interface, BCI). El objetivo del proyecto será comprobar si la evolución de Functional Link Networks con un enfoque multiobjetivo puede conseguir o no resultados competitivos en los tres dominios reales estudiados, centrándonos sobre todo en el dominio del BCI.

La memoria del proyecto está dividida en los siguientes capítulos:

- En el capítulo dos se explica cómo introducir no linealidades en el perceptron simple, mediante el uso de Functional Link Networks. Este capítulo contiene también la descripción del perceptron simple, así como las diferentes implementaciones de perceptrones que se han usado.

- En el tercer capítulo se trata la optimización multiobjetivo, comentando los principales conceptos de los problemas multiobjetivo, los algoritmos genéticos y la utilización de algoritmos genéticos multiobjetivo para resolver estos problemas, centrándonos especialmente en el NSGAII.

- El cuarto capítulo se centra en la interfaz cerebro-computadora con su definición y un resumen histórico del mismo, para ver los diferentes tipos de clasificación que se pueden realizar sobre el BCI.

- El quinto capítulo explica la idea desarrollada en este proyecto y las características importantes sobre el diseño general de este sistema.

- El sexto capítulo contiene toda la experimentación realizada sobre los diferentes dominios, así como los resultados que se obtienen para cada una de las pruebas realizadas.

- El séptimo capítulo trata las conclusiones obtenidas del trabajo realizado.

- El octavo capítulo comenta las líneas futuras que podrían desarrollarse para tratar de ampliar más el presente proyecto, y como podrían realizarse.

- El noveno capítulo es la bibliografía utilizada para realizar el proyecto.

Después se encuentran tres anexos al proyecto: en el primero se realiza un estudio sobre los datos del dominio BCI, el segundo anexo es un pequeño manual de usuario para poder utilizar el software realizado de una forma correcta y el tercer anexo es una gestión del proyecto en la que se comenta el método de trabajo realizado y un presupuesto del mismo.

2. INTRODUCCIÓN DE NO LINEALIDADES EN EL PERCEPTRON SIMPLE

En este capítulo se aborda la posibilidad de introducir no linealidades en el perceptron simple, por lo que comienza explicando el perceptron simple, además de las diferentes implementaciones de perceptrones que se han usado en este proyecto. Después tratamos la no linealidad con functional link networks (FLN) realizando una breve explicación sobre sus principales conceptos, así como su arquitectura.

2.1. Perceptron simple

El perceptron simple fue propuesto por Rosemblatt en 1958. Es la forma más simple de red de neurona, y se utiliza como un clasificador lineal en el que dado un conjunto de ejemplos o patrones se trata de determinar el hiperplano capaz de discriminar los patrones en dos clases.

La arquitectura del perceptron simple es la que se puede observar en la figura 1. Está formada por única neurona que recibe un número determinado de señales de entrada [$x_1, x_2, \ldots\ldots\ldots\ldots, x_n$]. Las conexiones llevan asociados unos pesos, que son números reales [$w_1, w_2, \ldots\ldots\ldots\ldots, w_n$]. Con todos estos datos se calcula la salida procesando la información recibida por las señales de entrada y siendo $y = f(x_1 w_1 + x_2 w_2 + ... + x_n w_w + u)$, u el umbral y f la función de activación, siendo para el perceptron simple la función la función de activación que viene dada por la siguiente ecuación y que se puede ver en la figura 2.

$$f(x) = \begin{cases} 1 & si\ x > 0 \\ 0 & en\ otro\ casc \end{cases}$$

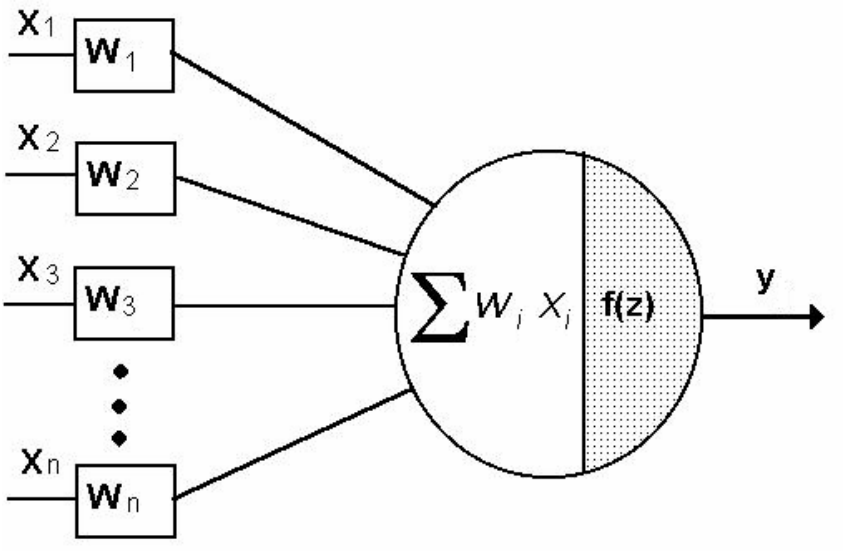


Ilustración 1. Arquitectura perceptron simple

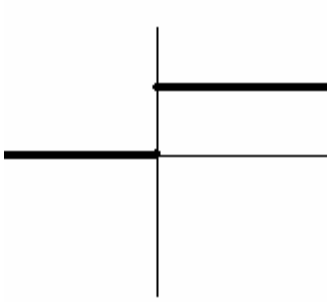


Ilustración 2. Función umbral

El proceso de aprendizaje del perceptron simple es un proceso iterativo supervisado, que se basa en la modificación de los parámetros de la red (pesos y umbral) hasta encontrar el hiperplano discriminante, siendo la condición de parada o bien un número de iteraciones definidas previamente o cuando el resultado se acerca en unos parámetros de error determinados.

El algoritmo del proceso de aprendizaje es el siguiente:

Paso 1: Inicialización aleatoria de los pesos y el umbral de la red

$w_i(0) \quad i = 0...n$

$u(0)$

Paso 2: Se toma un patrón entrada-salida $\left[x = (x_1,....., x_n), d(x)\right]$ siendo d(x) la salida deseada para ese patrón.

Paso 3: Se calcula la salida de la red: $y = f(x_1 w_1 + x_2 w_2 + ... + x_n w_n + u)$

Si $y = d(x)$ clasificación correcta.

Si $y \neq d(x)$ clasificación incorrecta, se modifican los parámetros.

Paso 4: Los pasos 2 y 3 se repiten para todos los patrones.
Paso 5: Los pasos 2, 3 y 4 se repiten hasta que se cumple la condición de parada.

Un posible criterio de parada podría consistir en tomar el número de aciertos para cada iteración del perceptron y si este número es igual al anterior se aumentarán las repeticiones, de manera que si se obtienen diez repeticiones se parará la ejecución de los perceptrones a pesar de que no se haya llegado al número de iteraciones definidas como máximo. En caso de que el número total de aciertos en esa iteración fuera mayor que el guardado, entonces guardaríamos ese valor y los pesos y el umbral de cada uno de los dos perceptrones y las repeticiones se inicializarían a cero.

De esta forma, nos aseguramos que los perceptrones no ejecutan iteraciones innecesarias que ralentizarían mucho la ejecución y además también nos aseguramos que tenemos el resultado de la mejor ejecución para después poder calcular los aciertos en test y validación con estos datos, de manera que obtenemos siempre el

mejor resultado de todos los encontrados en la ejecución de los dos perceptrones durante las iteraciones que se hayan ejecutado.

Para modificar los parámetros se utiliza la Ley de aprendizaje:

$$w_i(t+1) = w_i(t) + x_i d(x) \quad \forall \text{ i} \qquad u(t+1) = u(t) + d(x)$$

También se puede utilizar la regla de aprendizaje de Widrow-Hoff (1960), que tiene un comportamiento parecido a la ley anterior, pero que se basa en la idea de utilizar el error cometido por la red para adaptar mejor los pesos:

$$w_i(t+1) = w_i(t) + (x_i d(x) - y(x)) \quad \forall \ i \qquad u(t+1) = u(t) + (d(x) - y(x))$$

Otra idea que también se puede utilizar para evitar que los nuevos parámetros no clasifiquen mal patrones que antes estaban bien clasificados es utilizar la razón de aprendizaje, de manera que siendo $0<\alpha<1$ la razón de aprendizaje las leyes son:

$$w_i(t+1) = w_i(t) + \alpha\, d(x) x_i \quad \forall \text{ i} \qquad u(t+1) = u(t) + \alpha\ d(x)$$

Con la inclusión de la razón de aprendizaje se trata de controlar la brusquedad de las modificaciones de los parámetros. También es posible encontrarnos la regla de aprendizaje de Widrow-Hoff combinada con la razón de aprendizaje.

2.2. Diferentes implementaciones del perceptron simple para problemas de clasificación

El perceptron simple es una de las técnicas que se usa con más frecuencia en problemas de clasificación. La arquitectura descrita anteriormente sólo se puede utilizar para problemas de clasificación con dos clases. En nuestro caso, se abordan problemas de clasificación con más de dos clases (domino BCI que posee tres clases), por lo que resulta imposible utilizar un perceptron simple de los explicados anteriormente. Como solución a este problema se han decidido utilizar diferentes tipos de combinaciones de perceptrones simples para tratar el problema de la clasificación de tres clases diferentes y que se explican a continuación.

2.2.1. Utilización de dos perceptrones simples

La primera idea que puede surgir cuando se quiere resolver un problema de clasificación con tres clases distintas con perceptrones simples, consiste en la utilización de dos perceptrones simples de manera que el primero separe linealmente una clase de las otras dos y el segundo separe las otras dos clases entre ellas. La idea parece sencilla, y de hecho lo es, ya que codificando las clases a discriminar mediante dos bits (00, 01 y 10), conseguimos que los dos perceptrones actúen de la manera que queríamos.

Los perceptrones simples se entrenaran de forma independiente, es decir, el primero usará como clase a aprender el primer bit, y el segundo, el segundo bit. De la misma manera, para clasificar un patrón, se obtendrán los dos bits de la clase a partir de los perceptrones correspondientes.

Ambos perceptrones tendrán una función de activación umbral como la explicada en el apartado anterior, se trata por tanto de dos perceptrones simples combinados como se puede ver en la figura 3.

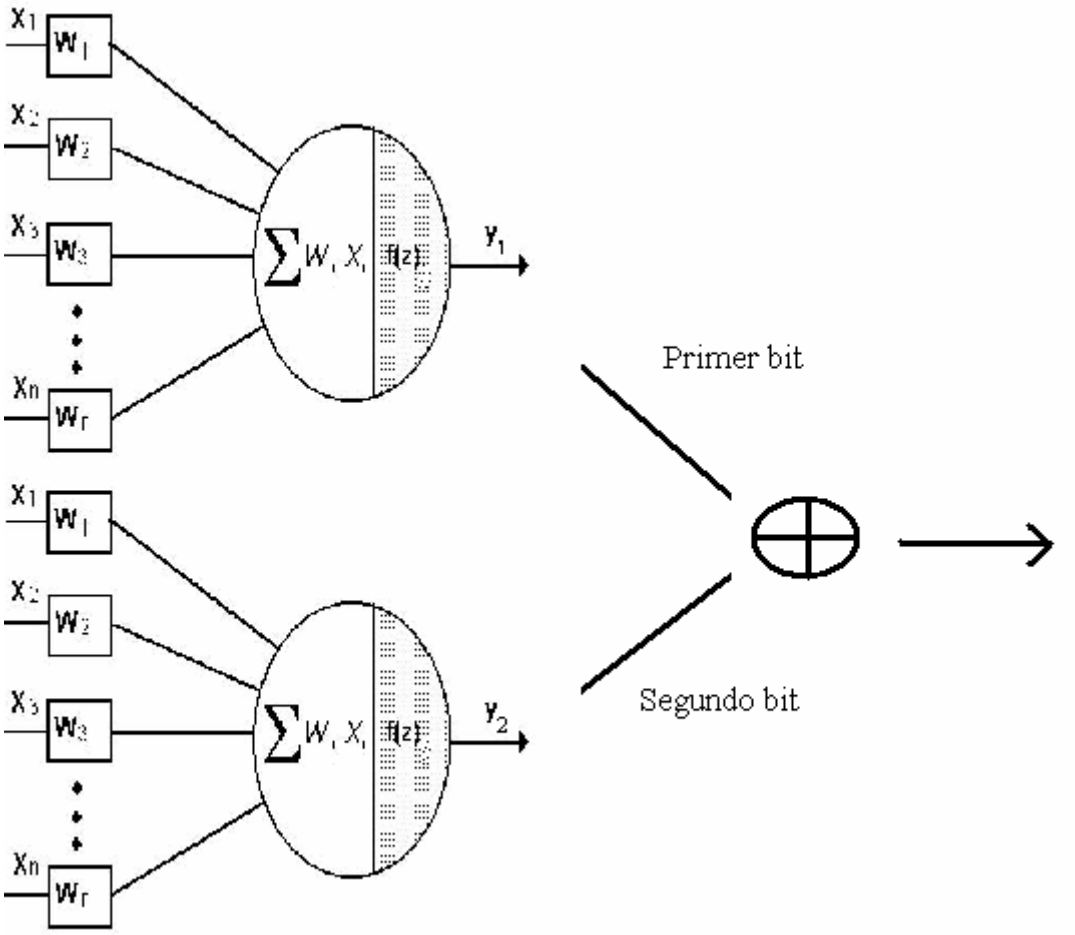


Ilustración 3. Arquitecto 2 PS combinados

2.2.2. Utilización de tres perceptrones simples (ONE-AGAINST-ALL)

Otra idea que también puede ser adecuada para la resolución de problemas de clasificación con tres clases pudiera ser la utilización de tres perceptrones simples, de manera que cada uno de ellos sirva para discriminar a una de las clases, y se eviten así los problemas que podrían surgir de la separación de una clase de las otras dos.

La idea sería utilizar tres valores binarios para guardar los valores de las clases, que serían [100, 010, 001] de manera que para cada una de las clases tenga un valor definido. En este caso, se combinan los tres perceptrones simples utilizando la regla de aprendizaje de Widrow-Hoff combinada con la razón de aprendizaje α.

Para comprobar si un ejemplo determinado es correcto o no, el conjunto de los tres perceptrones lo clasificará con la clase del perceptron que proporcione el mayor valor numérico. En la figura 4 se muestra la arquitectura antes explicada.

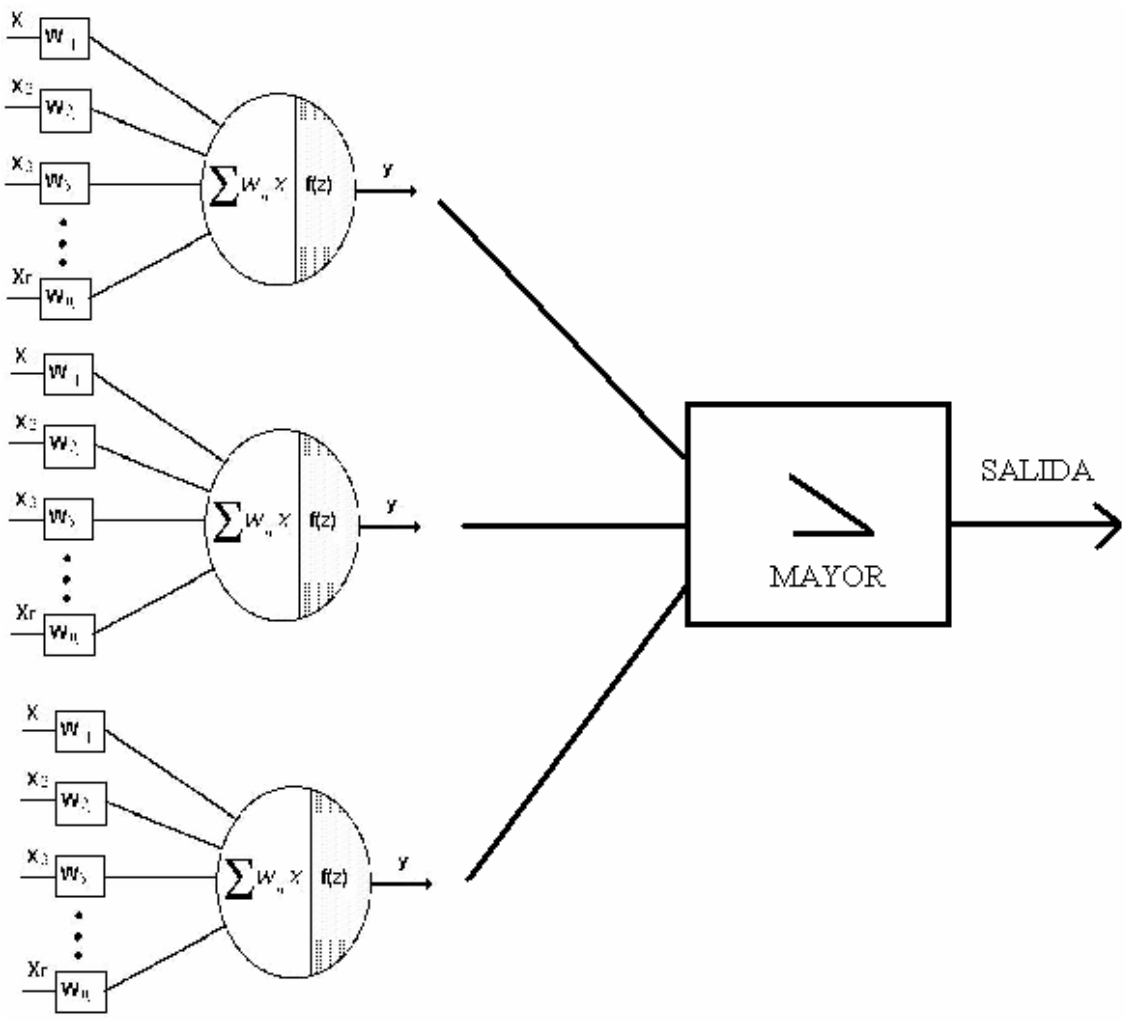


Ilustración 4. Arquitecto 3 PS combinados

2.2.3. Utilización de un sólo perceptron simple

Otra manera de abordar un problema de clasificación en tres clases con el perceptron simple, es codificar las clases como {0, 0.5, 1} y utilizar como función de activación de la neurona de salida, la función sigmoidal entre 0 y 1:

$$f(x) = \frac{1}{1+e^{-x}}$$

Al considerar esta función de activación, la regla de aprendizaje del perceptron simple cambia, pues es necesario derivar el error en la salida de la red. Llamando a $f'(x) = f(x)(1 - f(x))$ entonces la regla de aprendizaje sería la siguiente:

$$w_i(t+1) = w_i(t) + \alpha(d(x) - f(x))x_i f'(x) \quad \forall\ x_i \qquad y$$

$$u(t+1) = u(t) + \alpha \ (d(x) - f(x)) f'(x)$$

La forma de interpretar la salida de la red es: 0 si el valor de la función está comprendido entre [0, 0.33); 0.5 si está entre [0.33, 0.66) y 1 si es mayor o igual que 0.66. De manera que si salida coincide con la clase del ejemplo, entonces se tratará de un acierto.

Utilizando un sólo perceptron se pretende comprobar si a pesar de utilizar una función sigmoidal el tiempo de ejecución al no tratarse de varios perceptrones resulta ser menor, sin que por ello que los resultados sean peores.

2.3. Functional Link Networks (FLN)

Muchas aplicaciones de procesado de datos tales como procesado de imágenes y control en tiempo real, requieren una eficiente y rápida aproximación de funciones incluso en los casos dónde la expresión analítica de la función no existe pero se conocen algunos datos. Las redes neuronales artificiales son un buen procedimiento para resolver el problema propuesto. La razón fundamental se encuentra en su capacidad de aprender de la experiencia. Ahora bien, si se decide usar las redes neuronales artificiales se tendrá que elegir el tipo de red más apropiado a cada tipo de problema.

Una arquitectura sencilla como es el perceptron simple que se ha explicado en los apartados anteriores puede ser adecuada para resolver un problema lineal, pero sin embargo si en la solución del problema no existe un hiperplano, es decir, se trata de un problema no lineal, entonces el perceptron simple no es capaz de encontrar una solución.

Para solventar esta limitación nació el perceptron multicapa, que fue propuesto por Rumelhart, Hinton y Williams en 1986 y que nació con la idea de solventar el problema de la no linealidad en el perceptron simple. El perceptron multicapa tiene una arquitectura en las que sus neuronas se encuentran agrupadas en capas y cada neurona en cada capa está conectada a todas las neuronas de la siguiente capa, de manera que cada neurona procesa la información recibida y la respuesta se propaga a través de la conexión correspondiente y actúa como entrada para todas las neuronas de la siguiente capa. La figura 5 muestra la arquitectura del perceptron multicapa:

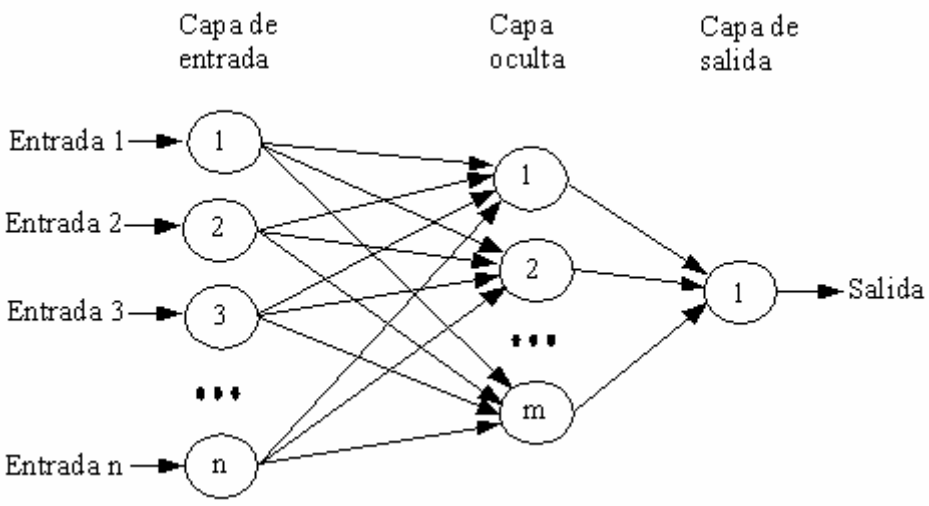


Ilustración 5. Arquitecto perceptron multicapa

Tanto el número de capas ocultas como el número de neuronas en cada capa puede variar de un problema a otro y en cuanto al aprendizaje de las neuronas se utiliza el algoritmo de retropropagación [Michie94].

Restringiéndonos al aprendizaje supervisado, se puede considerar que el perceptron multicapa es el más usado debido a que su arquitectura es capaz de aproximarse en teoría a cualquier función que se necesite [Hor91], pero problemas como que su razón de aprendizaje es muy baja y la cantidad de tiempo necesaria para realizar un entrenamiento óptimo hace que en ciertos problemas de la vida real este tipo de arquitectura no sea la más adecuada.

Para solventar estos inconvenientes del perceptron multicapa surgieron diversas técnicas como los functional link networks [Pao89] o los "higher order Networks" (HON's) [Gho92]. La idea básica de los functional link networks consiste en eliminar la capa oculta de los perceptrones multicapa e introducir no linealidad en un perceptron simple mediante el uso de atributos no lineales combinados con los atributos originales, de forma que todos ellos se presentan como atributos de entrada independientes.

Los functional link networks son mucho más modestos que el perceptron multicapa, ya que no garantizan una aproximación universal [Pao94]. Por tanto, para que un FLN funcione bien en un problema es necesario proporcionarle atributos no lineales apropiados para ese problema. Por otro lado, tienen la ventaja de que el algoritmo de entrenamiento es rápido, y se reduce a entrenar un perceptron simple, dado que la técnica de FLN encontrará la solución óptima pero en un espacio de búsqueda universal más reducido que el perceptron multicapa.

Una manera típica de construir atributos no lineales es mediante productos de los atributos originales (polinomios). Por ejemplo, para un problema con dos atributos originales y usando polinomios de grado dos, el FLN tendría como entradas {x1, x2, x1x2} de la forma que se observa en la figura 6, donde: $y = w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + u$.

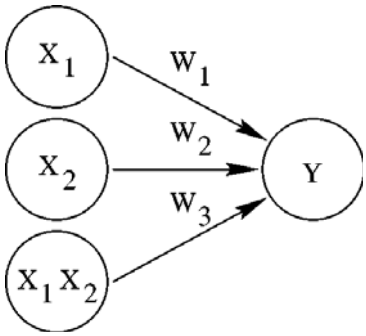


Ilustración 6.Arquitectura de un functional link simple bidimensional con dos atributos

Este conjunto de funciones no lineales en la entrada, si son linealmente independientes, incrementarán la dimensión del espacio de búsqueda en la salida y se obtendrán de manera rápida hiperplanos separadores en el espacio de patrones [Uge97].

Según los resultados obtenidos por Sobajic [Sob88], siempre es posible resolver con éxito mediante un perceptron simple con functional link la aproximación a funciones. Además con los resultados de Ugena [Ugena97] se comprueba como las redes ganan en simplicidad y se reduce además el tiempo para encontrar una solución óptima en comparación con los perceptrones multicapa.

Aunque los atributos no lineales se pueden construir de diversas formas, en este proyecto se han utilizado polinomios de grado dos, de manera que si los atributos originales de entrada fueran por ejemplo X, Y y Z, habría que añadir los atributos no lineales X*Y, X*Z y Y*Z. El objetivo del FLN sería encontrar una separación lineal de los datos en el nuevo espacio de entrada {X, Y, Z, X*Y, X*Z, Y*Z}, o expresado de otra forma es necesario encontrar la superficie de separación y+Ax+Bz+Cxy+Dxz+Eyz+F=0 que separe los datos de la mejor manera posible en el nuevo espacio de entrada. Esta superficie es una cuádrica en el espacio original {X, Y, Z}. En la figura 7 se pueden ver ejemplos de cuádricas en un espacio tridimensional.

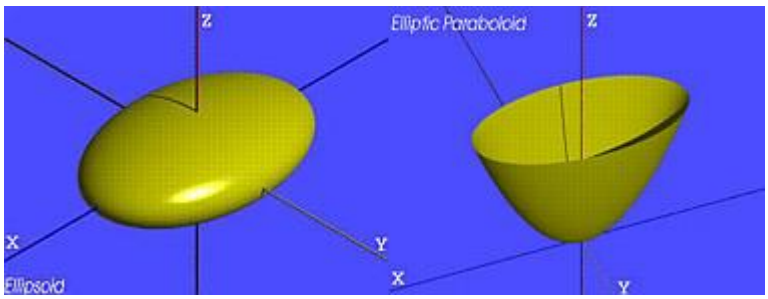


Ilustración 7.Cuádrica

El grado de un polinomio o de una función polinomial es la potencia del término que posee el exponente mayor. En el caso de los functional link el grado polinomial vendrá dado por el número de términos que intervengan en la operación, de esta manera con una cantidad de términos d y un grado polinomial de r, el número de términos a considerar es:

$$\sum_{i_1=1}^{d} \sum_{i_1=i_2}^{d} \cdots \sum_{i_r=i_{r-1}}^{d} x_{i_1} x_{i_2} \ldots x_{i_r}$$

La principal dificultad puede consistir en elegir un adecuado grado polinomial para cada tipo de problema, debido a que un alto grado polinomial utilizando todos los atributos daría lugar a un modelo muy complejo, ya que el número de entradas sería enorme en determinados dominios, y por ello elegir el grado polinomial es de gran importancia. Sin embargo, en algunos trabajos se muestra que polinomios de grado dos funcionan bien en varios dominios de Benchmark [Sierra01].

3. OPTIMIZACIÓN MULTIOBJETIVO

Este capítulo trata sobre la optimización multiobjetivo, y para ello en primer lugar se hablará sobre los problemas multiobjetivo y sus principales características, para continuar con una breve explicación de los algoritmos genéticos y después tratar sobre la utilización de estos algoritmos en la resolución de los problemas multiobjetivo. Y por último centrarse en un algoritmo concreto de optimización multiobjetivo, como es el NSGAII.

3.1. Problemas multiobjetivo

Para muchos campos dentro de las ciencias, resolver un problema o tomar una decisión significa hallar la mejor solución común a un conjunto de relaciones (tal es el caso de la economía, el diseño en ingeniería, situaciones de la vida cotidiana, etc.). Un ejemplo sencillo de esto se presenta al tratar de determinar las proporciones de ciertos componentes en un medicamento tomando en cuenta los menores costos de producción cuando a la vez se busca que éste no reduzca su eficacia y sea llamativo para una mayor cantidad de consumidores. Otro problema se presenta también si se requiere tomar la decisión de usar o no algún tipo de edulcorante en este medicamento, dado que esto podría afectar en mayor o menor medida a los aspectos anteriores. Muchos ejemplos se encuentran en el diseño y construcción de máquinas o piezas en la industria, en donde se deben tomar en cuenta diferentes características referentes a un mismo producto. Si cada una de estas relaciones puede expresarse como una función matemática, al referirnos a la mejora en conjunto, decimos que se deben optimizar todas las funciones de manera simultánea, definiéndose entonces un problema de optimización multiobjetivo.

La optimización multiobjetivo se puede definir como:

"Encontrar un vector de variables de decisión que satisfaga las restricciones dadas y optimice un vector de funciones cuyos elementos representan las funciones objetivo. Esas funciones forman una descripción matemática de los criterios a optimizarse y generalmente se encuentran en conflicto entre sí. Por lo tanto, el término optimizar significa encontrar las soluciones que darían valores aceptables para todas las funciones objetivo" [Osyczka85].

En términos matemáticos, el problema de optimización multiobjetivo, puede establecerse de la siguiente forma:

Encontrar un vector $x^* = [x_1^*, x_2^*, ..., x_n^*]^T$, que satisfaga las m restricciones:

$$g_i(x) \geq 0 \qquad i = 1,2,.....,n$$

y las p restricciones:

$$h_i(x) = 0 \qquad i = 1,2,....., p$$

y optimice la función vectorial

$$f(x) = [f_1(x), f_2(x),...., f_k(x)]^T$$

donde $x = [x_1, x_2,...., x_n]^T$ es el vector de variables de decisión.

La resolución de un problema de optimización multiobjetivo puede plantearse como la búsqueda de un vector que representa el conjunto de variables de decisión y el cual optimiza (maximiza o minimiza) en conjunto a las funciones objetivo. En la mayor parte de los casos dichas funciones pueden estar en conflicto unas con otras.

3.1.1. Pareto

Vilfredo Federico Damaso Pareto realizó importantes contribuciones al estudio de la economía y de la sociología, especialmente en el campo de la distribución de la riqueza y el análisis de las elecciones individuales. Fue el creador del concepto eficiencia de Pareto, y contribuyó, con ideas como la de la curva de indiferencia, al desarrollo de la microeconomía. Actualmente se utilizan muchas de sus contribuciones en el campo de la optimización multiobjetivo.

Ilustración 8. Pareto

3.1.2. Dominancia de Pareto

El concepto dominancia de Pareto se utiliza frecuentemente para comparar dos soluciones y decidir si una domina a la otra o no. Una solución se dice que domina a otra si cumple (para el caso de minimización):

Dado un vector $u = (u_1,....,u_k)$ *, se dice que domina a otro vector* $v = (v_1,....,v_k)$ *si y sólo si:*

$$\forall i \in \{1,...,k\}, u_i \le v_i \quad y \quad \exists i_0 \in \{1,....,k\} \mid u_{i_0} < v_{i_0}$$

3.1.3. Frente de Pareto

La respuesta al problema de hallar las mejores soluciones en un problema multiobjetivo consistirá en encontrar los puntos no dominados entre las soluciones encontradas, que se define como el conjunto solución del problema.

El conjunto de valores de la función objetivo con dominio restringido a los vectores del conjunto solución (es decir, los vectores no dominados) es lo que se conoce como frente de Pareto, y que se puede observar más claramente en la ilustración 9, de manera que se puede ver como los puntos P1 y P2 dominan al punto P3.

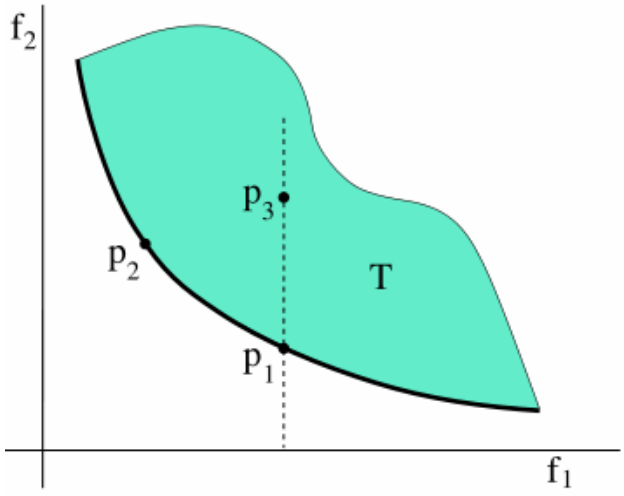


Ilustración 9. Frente de Pareto

3.1.4. Óptimo de Pareto

En problemas multiobjetivo, cuando los objetivos se encuentran en conflicto entre sí, se buscan normalmente soluciones compromiso en vez de una única solución. Por lo tanto, la noción de óptimo es diferente en estos casos. Dicha noción es comúnmente conocida bajo el término de óptimo de Pareto, como se puede observar en la ilustración 10 para el punto P1, ya que es el que domina a todos los demás puntos existentes.

Una solución x^* se *dice que es Pareto-óptima si y sólo si no existe otro vector* X *tal que* $v = f(x) = (v_1,....,v_k)$ *domine a* $u = f(x^*) = (u_1,....,u_k)$.

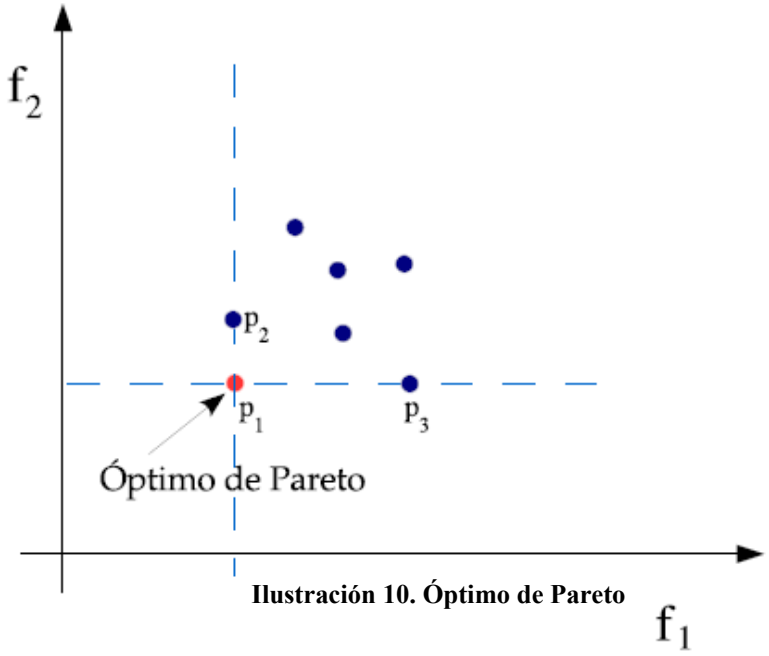


Ilustración 10. Óptimo de Pareto

3.1.5. Objetivo de la optimización multiobjetivo

Cuando existe un problema con dos o más objetivos en conflicto, generalmente el conjunto de soluciones óptimas contienen más de un elemento, por lo tanto, la tarea de elegir una solución entre el resto se vuelve una tarea difícil si no se cuenta con información adicional del problema. En estos casos, todas las soluciones que representan un óptimo de Pareto tienen la misma importancia, por lo que es necesario encontrar tantas soluciones como sea posible, que sean óptimos de Pareto. Por lo tanto, se puede llegar a la conclusión de que existen dos propósitos en la optimización multiobjetivo:

1. Encontrar un conjunto de soluciones lo más cercanas posible al conjunto de óptimos de Pareto real.
2. Encontrar un conjunto de soluciones lo más diversas posible. Es decir, que abarquen todo el frente de Pareto.

Aunque en la mayor parte de las ocasiones el frente óptimo no se conoce, si se puede establecer una relación entre el conjunto de soluciones encontradas y sus características, que serían las siguientes:

1. Encontrar un conjunto de soluciones que minimicen los objetivos lo mejor posible.
2. Encontrar un conjunto de soluciones lo más diversas posible.

En la ilustración 11 se clasifican las soluciones como deseables y no deseables tomando en cuenta qué tan cercanas se encuentran al frente de Pareto verdadero del problema de optimización.

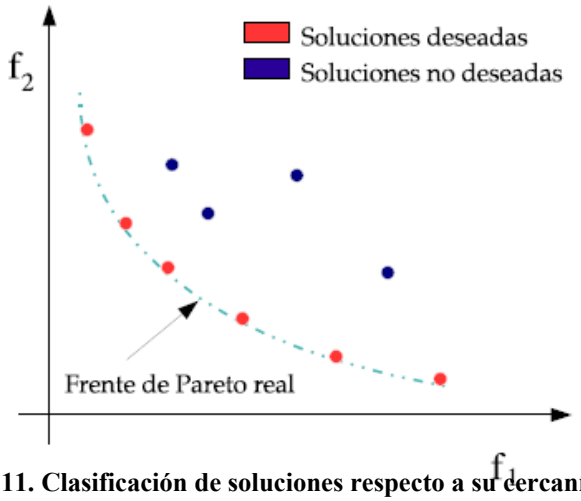


Ilustración 11. Clasificación de soluciones respecto a su cercanía al frente de Pareto

Por otro lado, el segundo punto es una característica deseada de manera particular para problemas multiobjetivo. Se desea obtener soluciones que además de pertenecer al conjunto de óptimos de Pareto, deben también encontrarse distribuidas lo más uniformemente posible. De esta forma se puede asegurar que se tiene un buen conjunto de soluciones.

En la ilustración 12 se observan dos conjuntos de soluciones; las que se encuentran mejor distribuidas son preferidas sobre aquellas que se encuentran muy cercanas entre sí, inclusive cuando ambos conjuntos de soluciones están sobre el frente de Pareto

.

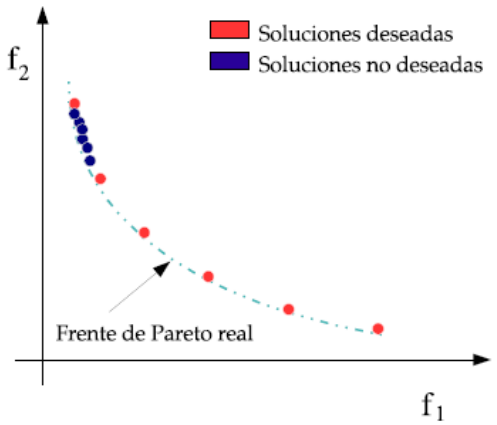


Ilustración 12. Clasificación de soluciones respecto a su distribución en el frente de Pareto

3.2. Algoritmos genéticos

En computación evolutiva se cuenta con diversas ramas y en años recientes se ha utilizado el término algoritmos evolutivos para englobar los principales tipos de técnicas contenidas en la computación evolutiva. Dichos algoritmos son los algoritmos genéticos, las estrategias evolutivas, los enjambres de partículas, la programación genética, entre otros.

A continuación, nos centraremos en los algoritmos genéticos, ya que las técnicas empleadas en este proyecto se basan en estos algoritmos evolutivos

3.2.1. Antecedentes históricos

Los algoritmos genéticos tienen sus antecedentes en la biología y comienzan con Darwin, que con su libro *El origen de las especies por medio de la selección natural o la preservación de las razas favorecidas en su lucha por la vida* [Darwin1859], nos habla sobre los principios de la selección natural.

Ilustración 13. Charles Darwin

Los principios de la selección natural se pueden resumir en cuatro proposiciones:

1. En cualquier generación, no todos los individuos de una especie logran reproducirse.
2. Los miembros de la especies no son idénticos; tienen variaciones individuales.
3. La mayoría de las variaciones son heredadas y transmitidas de los padres a los descendientes.
4. El éxito reproductivo no es aleatorio. Está asociado con las características heredadas; algunas de ellas son más beneficiosas que otras en situaciones particulares. Una de las principales deficiencias del argumento de Darwin es que, a pesar de que la herencia juega un papel preponderante en su teoría, no ofrece una explicación acerca de su funcionamiento.

Más tarde, los mecanismos de la herencia serían explicados en los trabajos realizados por Weismann y Mendel. El biólogo alemán August Friedrich Leopold Weismann establece la teoría del germoplasma [Weismann1893], que afirma que los organismos multicelulares están constituidos por células germinales (germoplasma) que contienen la información de la herencia, y células somáticas (somatoplasma) que se encargan de las funciones corporales.

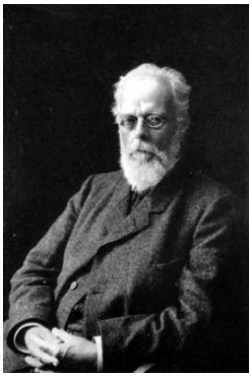


Ilustración 14. Weismann

Por su parte, el monje austriaco Johann Gregor Mendel publicó el trabajo *Experimentos de hibridación en plantas* [Mendel1901]. Como resultado de sus experimentos, Mendel estableció las leyes fundamentales de la genética:

- La primera de ellas, la ley de segregación, sostiene que los genes recibidos de los padres se segregan (separan) durante la formación de gametos sin afectarse entre sí.

- La segunda ley, llamada de independencia, sostiene que los pares de alelos se independizan durante la formación de gametos.

- La tercera, la ley de la uniformidad, sostiene que cada característica heredada se determina mediante dos factores provenientes de ambos padres, lo cual decide si un gen determinado es dominante o recesivo.

Ilustración 15. Johan Gregor Mendel

3.2.2. Características

Las principales características de los Algoritmos Genéticos son:

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas.
- Son algoritmos de búsqueda múltiple, luego dan varias soluciones.
- Son algoritmos que hacen una barrida mayor del subespacio de posibles soluciones válidas.
- Es una búsqueda paramétricamente robusta. Por lo tanto, tiende a converger a medida que se realizan las generaciones.

3.2.3. Representación

Se denomina cromosoma a una estructura de datos que contiene una cadena de variables de diseño de algún problema. Normalmente es una cadena binaria, pero también es posible usar una cadena con valores enteros o reales.

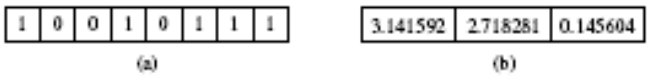


Ilustración 16.Ejemplo de un cromosoma binario (a) y otro real (b).

Llamamos gen a una subcadena del cromosoma que codifica, comúnmente, a un solo parámetro de diseño. Estos genes toman ciertos valores, llamados alelos, de algún alfabeto genético. De esta manera, si usamos una representación binaria, los alelos pueden tomar el valor de 0 o de 1. Se denomina genotipo a la codificación de los parámetros de diseño. Mientras que fenotipo es la decodificación del genotipo, con el fin de obtener los valores de los parámetros usados como entrada en la función objetivo del problema que se trata.

Ilustración 17. Decodificación del genotipo al fenotipo.

Un individuo es una solución potencial al problema que se trata. Cada individuo contiene un cromosoma. A un conjunto de individuos se le denomina población. El fitness de un individuo es la evaluación de la función de evaluación e indica qué tan bueno es el individuo (es decir, la solución al problema) con respecto a los demás. La función de evaluación de un problema es, junto con el diseño del genotipo dos de las características más importantes a la hora de encontrar la mejor solución posible a cualquier problema.

3.2.4. Operadores evolutivos

Existe una amplia variedad de operadores evolutivos que se emplean en los algoritmos genéticos. A continuación se detallan los principales:

- Selección: La selección determina la probabilidad de elegir un individuo para que produzca descendencia por medio de la reproducción y la mutación.

- Reproducción. Los operadores de reproducción (cruce) tienen la finalidad de heredar la información (genes) de dos o más padres a la descendencia. La reproducción entre cromosomas se realiza intercambiando segmentos de cadenas lineales de longitud fija.

- Mutación. La mutación se consigue realizando pequeñas modificaciones al cromosoma de un individuo.

3.2.5. Algoritmo

Desarrollado por John H. Holland [Holland75], el algoritmo genético opera entonces a nivel de genotipo de las soluciones mediante la siguiente secuencia:

I Comenzar con una población inicial, la cual puede ser generada de manera aleatoria.
II Calcular el fitness de cada individuo.
III Aplicar el operador de selección con base en el fitness de la población.
IV Aplicar los operadores genéticos de cruce y mutación a la población actual para con estos generar a la población de la siguiente generación.
V Ir al paso II hasta que la condición de parada se satisfaga.

Al igual que en muchas otras heurísticas, el comportamiento del algoritmo genético es altamente dependiente de los parámetros iniciales (tamaño de la población, porcentaje de cruce, porcentaje de mutación, número de generaciones, etc.), por lo que será necesario ajustar esos parámetros para tratar de mejorar la solución para los objetivos del problema.

3.3. *Aplicación de* algoritmos genéticos a la optimización multiobjetivo

3.3.1. Antecedentes históricos

Hoy en día existen más de treinta técnicas de programación matemática para resolver problemas de optimización multiobjetivo. Con todo, la complejidad de muchos problemas de optimización multiobjetivo del mundo real vuelve a estas técnicas inadecuadas o incluso inaplicables para resolverlos. La complejidad de estos problemas se debe, por ejemplo: a la multimodalidad, a la alta dimensionalidad del espacio de búsqueda, a la discontinuidad de sus funciones objetivo, a desconexiones tanto en el espacio de las variables de decisión como en el de las funciones objetivo, o a que son NP-completos.

Algunos investigadores [Coello99, Deb99, Fogel99] han identificado algunas dificultades que tienen las técnicas clásicas para resolver problemas de optimización multiobjetivo. A continuación se detallan algunas de ellas:

1. Los algoritmos se necesitan ejecutar varias veces para encontrar varias soluciones del conjunto de óptimos de Pareto.
2. La mayoría de los algoritmos requieren información sobre el dominio del problema que se trata.
3. Algunos algoritmos son sensibles a la forma o continuidad del frente de Pareto.
4. En los problemas que involucran incertidumbre o eventos estocásticos, los métodos clásicos son inadecuados.
5. La dispersión de las soluciones del frente de Pareto depende de la eficiencia del optimizador monoobjetivo.

La complejidad de los problemas de optimización multiobjetivo del mundo real ha conducido a la búsqueda de enfoques alternativos para resolver este tipo de problemas. Uno de esos enfoques lo encabezan los algoritmos evolutivos.

A finales de 1960, Rosenberg [Rosenberg67] plantea utilizar un método genético de búsqueda para resolver problemas de optimización multiobjetivo. No obstante, no fue hasta 1984 cuando David Schaffer [Schaffer84] propone la primera implementación de lo que actualmente conocemos como algoritmo evolutivo multiobjetivo. A partir de ese momento, varios investigadores han desarrollado su propio algoritmo evolutivo multiobjetivo. La publicación de los resultados de estos algoritmos mostró la superioridad de éstos sobre las técnicas clásicas de programación matemática.

Los algoritmos genéticos son adecuados para resolver problemas de optimización multiobjetivo gracias a que trabajan simultáneamente con un conjunto de soluciones potenciales (es decir, la población). Esta característica les permite encontrar varias soluciones del conjunto óptimo de Pareto en una sola ejecución. Asimismo, son menos sensibles a la forma o continuidad del frente de Pareto. En general, la principal característica de un algoritmo genético multiobjetivo es que es capaz de mantener un conjunto de soluciones potenciales, las cuales son sometidas a un proceso de selección y es manipulado por operadores genéticos, generalmente la reproducción y la mutación.

3.3.2. Algoritmos genéticos aplicados a la resolución de problemas multiobjetivo

Como ya hemos comentado, las técnicas evolutivas son particularmente atractivas para el tratamiento de problemas multiobjetivo, ya que la solución de un problema multiobjetivo no se resume a un único resultado óptimo, sino que consiste de una colección de valores maximales a la vez; las técnicas evolutivas se acoplan bien en este aspecto ya que en ellas se procesa de manera simultánea una población de individuos, los mismos que representan soluciones a mejorar en cada iteración. Al final de la ejecución del algoritmo evolutivo se obtiene una aproximación al frente de Pareto.

Desde la primera implementación de un algoritmo evolutivo multiobjetivo se han propuesto una amplia variedad de algoritmos, que de forma general, los podemos dividir en: Funciones de agregación, Métodos poblacionales, Métodos basados en jerarquización de Pareto.

3.3.3. Funciones de agregación

Las funciones de agregación son quizá el método más simple para manejar múltiples objetivos y consiste en combinar todos los objetivos en uno solo usando operaciones aritméticas tales como la suma o multiplicación. Estas técnicas son conocidas como "funciones de agregación" porque combinan todos los objetivos del problema en uno solo. Las funciones de agregación pueden ser lineales o no lineales y ambos tipos han sido utilizados con éxito relativo en múltiples ocasiones. Sin embargo, han sido poco apreciadas por investigadores de optimización evolutiva multiobjetivo debido a sus limitaciones, ya que todo depende de la función de evaluación elegida y por tanto, no pueden generar muchos puntos, ni que éstos sean diversos respecto al frente de Pareto.

3.3.4. Métodos poblacionales

En este caso, con el fin de diversificar la búsqueda en un algoritmo evolutivo, se utiliza la población del mismo para explorar distintos tipos de soluciones, aunque el concepto de dominancia de Pareto no se encuentra directamente incorporado en el

proceso de selección. Un ejemplo clásico de este tipo de métodos es VEGA [Schaffer84].

VEGA consiste básicamente de un algoritmo genético simple con un mecanismo de selección modificado. En cada generación, se genera un número de subpoblaciones aplicando selección proporcional de acuerdo a la función objetivo en turno. De esta forma, para problemas con k objetivos, k subpoblaciones de tamaño M/k cada una, son generadas (asumiendo un tamaño de población total de M individuos). Estas subpoblaciones son mezcladas entre sí para obtener una nueva población de tamaño M, en la cual el algoritmo genético aplica los operadores de cruce y mutación. VEGA tiene varios problemas, de los cuales, el más serio es que su esquema de selección es opuesto al concepto de dominancia de Pareto. Por ejemplo, si un individuo codifica una buena solución compromiso para todos los objetivos, pero no es la mejor en ninguno de ellos, ésta es descartada. Evidentemente, dicho individuo debería de ser preservado porque codifica una solución del frente de Pareto.

Un aspecto interesante de VEGA es que continúa siendo utilizado por algunos investigadores principalmente porque es apropiado para problemas en los que necesitemos lidiar con un gran número de objetivos.

3.3.5. Métodos basados en jerarquización de Pareto

Los métodos basados en jerarquización de Pareto consisten en un esquema de selección basado en el concepto del óptimo de Pareto. Estos métodos pueden dividirse históricamente en dos generaciones. La primera generación se caracteriza por el uso de compartición de fitness y nichos combinados con un mecanismo de selección basado en el óptimo de Pareto. Los algoritmos más representativos de esta primera generación son los siguientes:

1. Non-dominated Sorting Genetic Algorithm (NSGA): Este algoritmo fue propuesto por Srinivas y Deb [Deb94]. Antes de realizar la selección, la población se ordena de acuerdo a su no dominancia, es decir, todos los individuos no dominados son clasificados en una categoría. Con el fin de mantener la diversidad de la población, estos individuos ya clasificados son mezclados con sus valores de fitness. A continuación, este grupo de individuos clasificados se ignoran de la población total para obtener la otra capa de individuos no-dominados. El proceso continúa hasta que todos los individuos de la población hayan sido clasificados, de manera que todos los individuos se van colocando en frentes. El mecanismo de selección utilizado es el método del sobrante estocástico. Debido a que los individuos del primer frente son los que obtienen el valor máximo de fitness, éstos son seleccionados en mayor proporción comparados con los demás individuos de la población. Esto permite guiar la búsqueda hacia regiones no dominadas y produce la convergencia de la población hacia dichas regiones. La mezcla ayuda a distribuir la población sobre el frente de Pareto del problema.

2. Niched-Pareto Genetic Algorithm (NPGA): Fue propuesto por Horn et al. [Horn94]. El NPGA usa un esquema de selección por torneo basado en la dominancia de Pareto. La idea básica del algoritmo es la siguiente: Dos individuos se seleccionan de forma aleatoria y se comparan contra un subconjunto de la población total (típicamente alrededor del 10% de la población). Si uno de ellos es dominado (por los individuos seleccionados aleatoriamente de la población) y el otro no, gana este último. Cuando

ambos competidores son dominados o no-dominados, el resultado del torneo se decide mediante compartición de fitness (es decir, el individuo que resida en la región menos poblada del espacio de búsqueda es el ganador).

3. Multiobjective Genetic Algorithm (MOGA): Fue propuesto por Fonseca y Fleming [Fonseca93]. En MOGA, la clasificación de un individuo determinado corresponde al número de individuos en la población actual que lo dominan. A todos los individuos que son no-dominados se les asigna el valor de uno en la clasificación. El fitness se mide de la siguiente manera:

a) Se ordena la población de acuerdo a su clasificación.
b) Se asigna el fitness a los individuos mediante la interpolación del mejor al peor de acuerdo a una función, generalmente lineal, aunque no necesariamente.
c) Se realiza un promedio de fitness con los individuos de la misma clasificación, de forma que todos ellos se muestreen con la misma proporción. Este procedimiento mantiene constante el fitness de la población global y a su vez mantiene una presión de selección apropiada, definida por la función utilizada.

La segunda generación de algoritmos genéticos multiobjetivo nació con la introducción de la noción de elitismo. En el contexto de optimización multiobjetivo, generalmente se utiliza el término elitismo para hacer referencia al uso de una población externa (también llamada población secundaria) con el fin de retener los individuos no-dominados. El uso de dicho archivo externo, despierta varias inquietudes, como son: ¿Cómo interactúa el archivo externo con la población principal? ¿Qué se debe hacer cuando el archivo externo se llene? ¿Se debe imponer un criterio adicional para agregar elementos al archivo en lugar de sólo utilizar la dominancia de Pareto?

Los algoritmos genéticos multiobjetivo más representativos de la segunda generación son los siguientes:

1. Strength Pareto Evolutionary Algorithm (SPEA): Este algoritmo fue introducido por Zitzler y Thiele [Zitzler99]. SPEA usa un archivo que contiene soluciones no dominadas previamente encontradas. En cada generación, individuos no dominados se copian a un conjunto externo. Para cada individuo en este conjunto externo, se calcula un valor de fuerza. Este valor es similar al valor de clasificación del MOGA, ya que es proporcional al número de soluciones a las que un determinado individuo domina. En SPEA, el valor de aptitud para cada miembro de la población en curso se calcula de acuerdo a las fuerzas de todas las soluciones externas no dominadas que dominan a dicho individuo.
2. Strength Pareto Evolutionary Algorithm 2 (SPEA2): Este algoritmo [Zitzler01] tiene tres principales diferencias respecto a su predecesor SPEA [Zitzler99].
a) Incorpora una estrategia de asignación de fitness de grano fino, la cual toma en cuenta, para cada individuo, el número de individuos que domina y el número de individuos por los cuales es dominado.
b) Usa la técnica de estimación de densidad del vecino más cercano, que guía la búsqueda de forma más eficiente.
c) Tiene un método mejorado de truncamiento del archivo externo que garantiza la preservación de soluciones de frontera.

3. Pareto Archived Evolution Strategy (PAES): Este algoritmo fue introducido por Knowles y Corne [Knowles00]. PAES consiste de una estrategia evolutiva del tipo (1+1) (o sea, un solo padre que genera un solo hijo) en combinación con un archivo histórico que guarda algunas de las soluciones no-dominadas encontradas previamente. Este archivo es usado como conjunto de referencia y cada individuo mutado es comparado contra dicho conjunto. Un aspecto interesante del algoritmo es el proceso para mantener diversidad que consiste en un procedimiento de densidad de población que divide el espacio de objetivos de forma recursiva.

4. Niched Pareto Genetic Algorithm 2 (NPGA 2): Erickson et al. [Eri01] propusieron una versión corregida del NPGA llamada NPGA 2. Este algoritmo utiliza jerarquización de Pareto pero mantiene la selección por torneo. En este caso no se utiliza un archivo externo y el mecanismo de elitismo es similar al adoptado en el NSGA-II. El conteo de nichos en el NPGA 2 es calculado usando individuos en la siguiente generación parcialmente llena en lugar de usar la generación actual.

5. Micro-Genetic Algorithm: Este método fue introducido por Coello y Toscano Pulido [Coello01]. Un microalgoritmo genético es un algoritmo genético con una población muy pequeña (no más de 5 individuos) y un proceso de reinicialización.

6. Non-dominated Sorting Genetic Algorithm II (NSGA-II): Deb et al. [Deb00] propusieron una versión mejorada del NSGA, y que aparece explicada con más detalle en el siguiente apartado.

3.3.6. Características de los algoritmos evolutivos multiobjetivo

Con base en la descripción anterior, las características que se demandan del algoritmo evolutivo multiobjetivo son las siguientes:

1. Tener baja complejidad computacional: El algoritmo híbrido final requeriría de varias ejecuciones del motor de búsqueda que se elija, por lo que se busca un algoritmo con una complejidad razonablemente baja.

2. Ser un algoritmo de la segunda generación: Además de buscar soluciones basadas en el frente de Pareto, el usar elitismo es una característica deseable para evitar desechar soluciones que nos sean útiles a lo largo de la búsqueda, por lo que se prefiere este tipo de algoritmos sobre los de primera generación.

3. Ofrecer soluciones óptimas y bien distribuidas: es necesario que las soluciones se encuentren lo más dispersas posibles, y además que sean buenas.

3.4. NSGA-II

El algoritmo NSGA-II (Non-dominated Sorting Genetic Algorithm, versión II) fue presentado por K. Deb y sus alumnos del Laboratorio de Algoritmos Genéticos del Instituto Tecnológico Kanpur en India en el año 2000 (Deb et al., 2000) [Deb00]. Surgió como una versión mejorada del algoritmo NSGA [Deb94], de quién heredó su estructura principal, pero incluyendo características distintivas para resolver tres aspectos fuertemente criticados en la comunidad de investigadores sobre el NSGA: el ordenamiento no dominado, la ausencia de elitismo y la dependencia del parámetro para

aplicar la técnica de sharing. Las características principales del algoritmo NSGA-II abarcan:

- El ordenamiento no-dominado elitista mediante una técnica de comparación que utiliza una subpoblación auxiliar, que le permite disminuir la complejidad de los chequeos de dominancia de O (MP^3) a O (MP^2), siendo M el número de funciones objetivo y P el tamaño de la población utilizada.
- La utilización de una técnica de crowding que no requiere especificar parámetros adicionales para la preservación de diversidad en la población, eliminando la dependencia de parámetros como el de sharing utilizado por el NSGA original.
- La asignación de valores de fitness en base a los niveles o rangos de no dominancia, se hereda del NSGA original, aunque se considera en el procedimiento de asignación los valores de distancia de crowding utilizados para evaluar la diversidad de las soluciones.

A continuación se presenta un esquema del algoritmo NSGA-II, basado en la descripción de Deb et al. (2000). Pueden apreciarse los operadores mencionados, utilizados para el ordenamiento no dominado, evaluación de la diversidad mediante la técnica de crowding y asignación de fitness.

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
mientras (no CriterioParada) hacer
    R = Padres ∪ Hijos
    Frentes = Sorting No Dominado(R)
    NuevaPop = ∅
    i=1
    mientras |NuevaPop| + |Frentes(i)| ≤ sizepop
        Calcular Distancia de Crowding (Frentes(i))
        NuevaPop = NuevaPop ∪ Frentes(i)
        i++
    Sorting por Distancia (Frentes(i))
    NuevaPop = NuevaPop ∪ Frentes(i)[1:(sizepop - |NuevaPop|)
    Hijos = Seleccion y Reproduccion(NuevaPop)
    generacion ++
    P(generacion) = NuevaPop
retornar Mejor Solucion Hallada
```

Tabla 1. Algoritmo NSGA-II

Una vez terminado el ordenamiento no dominado, la nueva población se llena con soluciones de diferentes frentes no dominados. El llenado comienza con el mejor frente no dominado y continúa con las soluciones del segundo frente no dominado, y así sucesivamente. Los frentes que no puedan entrar en la nueva población serán borrados. En caso de que el último frente permitido sea considerado, puede ocurrir que existan más soluciones en el último frente que el total de localidades disponibles en la nueva población. En este caso, en lugar de descartar de forma arbitraria a los miembros

sobrantes del último frente, se utiliza una estrategia de nichos para escoger los miembros del último frente que formarán parte de la nueva población; éstos serán los que residan en las regiones menos pobladas en dicho frente.

La medida de crowding se utiliza para seleccionar las soluciones más dispersas entre los individuos del último frente utilizado en la nueva población. Cuanto mayor sea la distancia de crowding de una solución al resto de su frente, mejor, ya que hay menos concentración en esa zona. El algoritmo NSGA-II utiliza esta distancia con la intención de obtener un frente lo más disperso posible.

4. INTERFAZ CEREBRO-COMPUTADORA

En este capítulo se habla sobre el brain computer interface (BCI) comenzando con una definición y un resumen histórico del mismo. Posteriormente se ofrece un punto de vista sobre los diferentes tipos de clasificación que se pueden realizar sobre los BCI's para especificar su funcionamiento y su principal clasificación.

4.1. Introducción

Cuando se oye hablar por primera vez del BCI es inevitable pensar que la comunicación telepática ha saltado del mundo de la ciencia ficción al mundo real, o que existen verdaderas "máquinas de la verdad" que pueden leer nuestros pensamientos.

Pero nada más lejos de la realidad, una interfaz cerebro computadora (BCI) es un sistema que trata de crear una comunicación directa entre el cerebro de los seres humanos o el de los animales con un dispositivo. Estos dispositivos externos, pueden transmitir o recibir señales desde y hacia el cerebro que pueden utilizarse para restaurar la función o movimiento a los órganos de los sentidos o las extremidades.

En general se puede afirmar que el público objetivo de los sistemas BCI son las personas afectadas por algún tipo de discapacidad motora grave, con aplicaciones que les permita la selección de letras con síntesis de voz para comunicarse con su entorno, el control de su silla de ruedas o del entorno domótico, e incluso el movimiento de una neuroprótesis.

Sin embargo, también el gran público puede beneficiarse de la tecnología BCI, siendo los videojuegos, y el uso de estos sistemas en entornos militares las principales aplicaciones.

Por tanto, el objetivo actual tanto de la industria como de la comunidad científica es incrementar de forma exponencial el desarrollo de las tecnologías BCI, para lo que ya se han puesto en marcha sitios Web informativos (como BCI-Info), implementaciones de código abierto (BCI2000, BioSig), y competiciones o eventos en los que los diferentes grupos de investigación comparan sus algoritmos de procesado de la señal sobre un conjunto de datos comunes.

Y es que las tecnologías BCI deben afrontar un gran número de problemas y limitaciones, entre otros los siguientes:

- Se trata de un campo de investigación multidisciplinar.
- El limitado público objetivo (colectivo de discapacitados) no promueve grandes inversiones por parte de la industria.
- Se generan falsas expectativas sobre las funcionalidades ofrecidas.
- Los sistemas BCI más adecuados para la comercialización (basados en EEG) son a la vez los menos precisos.
- Es necesario un periodo de entrenamiento con el sistema que puede durar incluso meses.

4.2. Antecedentes históricos

Las tecnologías BCI constituyen un área de investigación relativamente joven, a pesar de hace ya casi ocho décadas que Hans Berger consiguió registrar la actividad bioeléctrica cerebral mediante la electroencefalografía (EEG). Sin embargo no fue hasta la década de 1970 cuando comenzaron a surgir diferentes programas de investigación en torno a BCI, motivados entre otras razones por la observación científica de la correlación entre las señales de EEG y los movimientos reales (e incluso imaginados) de los usuarios, así como determinadas actividades mentales de éstos. Se realizaron numerosos experimentos con animales como ratas y monos, de manera que fueran capaces de realizar movimiento, se consiguió que los monos fueran capaces de mover cursores o mover brazos robóticos.

Ilustración 18. Prueba laboratorio BCI

El potencial médico de la tecnología BCI quedó patente a finales de los 90 mediante la implantación de un electrodo en el córtex motor de un paciente que presentaba parálisis por debajo de su cuello y había perdido la facultad del habla, de forma que el paciente era capaz de comunicarse moviendo un cursor en un ordenador. Desde entonces la investigación en las tecnologías BCI, aún requiriendo la colaboración de múltiples disciplinas (biotecnología, ingeniería biomédica, nanotecnología, ciencia del conocimiento, tecnología de la información, informática, neurociencia, matemática aplicada, etc.), ha experimentado un gran crecimiento.

De hecho, en el año 2001 nació un evento bianual, la competición BCI, en la que cualquier investigador podía demostrar la eficiencia de su sistema BCI (o parte de él) contra una serie de señales cerebrales proporcionadas por algunos de los más importantes grupos de investigación sobre BCI.

4.3. Clasificación BCI

Los sistemas BCI's pueden clasificarse en dependientes e independientes de acuerdo a la señal electrofisiológica que utilicen. Asimismo pueden también clasificarse de acuerdo a la tarea cognitiva o mental en pasivos y activos. Mientras que otra clasificación está en relación al tiempo de ejecución de la tarea que los subdivide en BCI's síncronos y asíncronos. Por último debemos destacar la clasificación en metabólicos y no metabólicos; como así mismo, los sistemas invasivos y no invasivos.

Los BCI's dependientes son aquellos que utilizan control ocular de la mirada. Sutter (1992) describió como un usuario puede seleccionar un símbolo de entre 64 propuestos en pantalla en una formación de 8X8, de modo que focalizándose en uno de ellos podía escribir entre 10 y 12 palabras por minuto Los símbolos van cambiando de color o

parpadeando con una cierta frecuencia, lo que induce un patrón espacio-temporal distintivo en la corteza visual del cerebro del usuario (potencial evocado). Dicho patrón depende de la frecuencia de parpadeo de la letra focalizada. Una desventaja es que este método requiere un control estable de los músculos oculares, ya que son necesarios para focalizar la letra. Sutter describe a este sistema como Brain Response Interface (BRI) utilizando un teclado de pantalla como interface.

Los BCI's que no dependen de ninguna actividad muscular se definen como Independientes. En este tipo BCI's el usuario no realiza ninguna actividad muscular, y por tanto será necesario que las instrucciones se obtengan por algún otro tipo de mecanismo, que normalmente serán mediante la colocación de aparatos en la parte del córtex cerebral del usuario, por ejemplo electrodos.

De las numerosas formas de clasificar los diferentes tipos de BCI, la principal es la siguiente:

4.3.1. BCI invasivas

Los Brain Computer Interface Devices son aquellos implantados directamente en el cerebro y obtienen la más alta calidad de las señales. Estos dispositivos se utilizan para proporcionar funcionalidad a las personas paralizadas. Las BCI's invasivas también pueden utilizarse para restaurar la visión del cerebro con el exterior y cámaras para restablecer el uso de las extremidades mediante el uso de un cerebro robótico que controle brazos y las piernas.

El problema con este tipo de dispositivo, es que el tejido cicatrizal que se forma puede causar al paciente una reacción al cuerpo extraño. Esto reduce su eficiencia y aumenta el riesgo para el paciente.

4.3.2. BCI Parcialmente invasivas

Las BCI's parcialmente invasivas, son implantados dentro del cráneo pero fuera del cerebro. Aunque la fuerza de la señal utilizando este tipo de dispositivo de BCI es un poco más débil, las BCI's parcialmente invasivas tiene menos riesgo de la formación de tejido cicatrizal.

4.3.3. BCI no invasivas

Las interfaces no invasivas, a pesar de que tienen la menor calidad de la señal cuando se trata de comunicarse con el cerebro (el cráneo, distorsiona la señal), es también la más segura porque no es necesario penetrar en el cerebro. Este tipo de dispositivo se ha diseñado para tener éxito en un paciente en que la capacidad de mover los músculos y los implantes sirvan para restablecer el movimiento de forma parcial.

A pesar de su corta historia como área de investigación, los sistemas BCI han atraído a muchos investigadores de diferentes disciplinas durante la última década con el objetivo común de desarrollar un interfaz hombre máquina fiable y eficiente controlado por las señales recogidas directamente del cerebro. No obstante, cada grupo

de investigación ha generado su propio sistema BCI, de forma que las diferentes tecnologías y diseños empleados hace prácticamente imposible establecer comparaciones directas entre unos y otros. Aún así, es posible describir a alto nivel los diferentes componentes funcionales que puede presentar un sistema BCI.

La figura 19 muestra el modelo funcional genérico al que podrían responder la práctica totalidad de los sistemas BCI, si bien muchos de ellos no integran todos los componentes o funciones recogidas en dicho modelo.

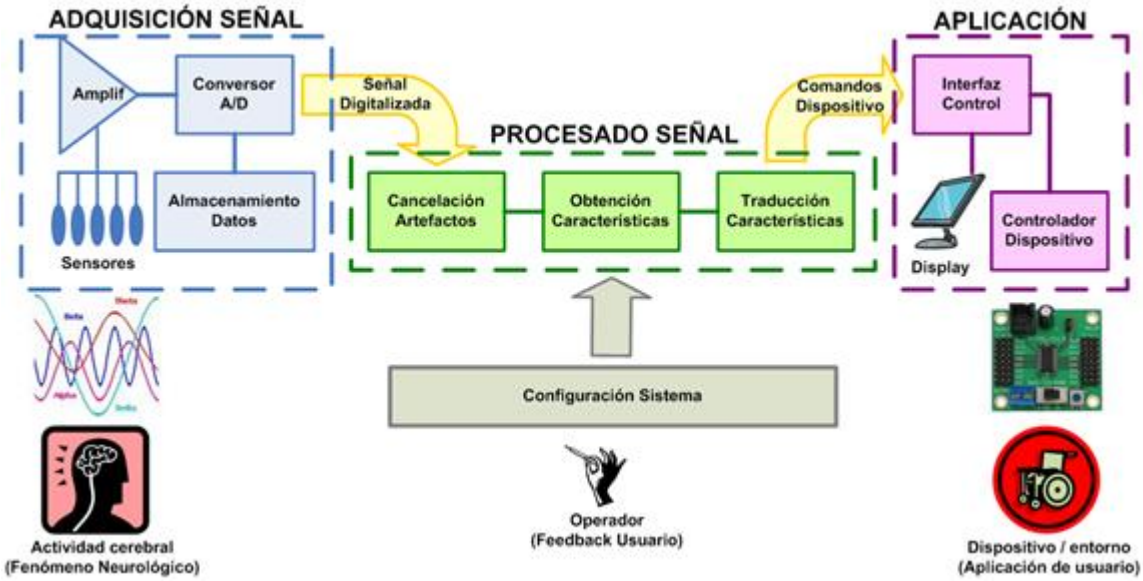


Ilustración 19. Modelo funcional sistema BCI

Se distinguen 4 bloques funcionales:

1. Adquisición de señal: cuyo objetivo es el registro de la actividad cerebral del usuario y su adecuación al bloque de procesado de señal. Se trata por tanto de capturar el fenómeno neurológico que refleja las intenciones del usuario mediante sensores (electrodos en el cuero cabelludo) y preparar la señal registrada para su procesado posterior mediante etapas de amplificación y digitalización. Aunque para el procesado en tiempo real y, en consecuencia, para el funcionamiento del sistema BCI no resulta necesario almacenar la señal registrada, casi todos los sistemas BCI incorporan esta etapa con objeto de permitir posteriores análisis y procesados de la misma (por ejemplo utilizando algoritmos de procesado diferentes).
2. Procesado de señal: que recibe la señal digitalizada y la transforma en los comandos que entiende el dispositivo sobre el que usuario está actuando. Este bloque funcional se divide en tres etapas que actúan de forma secuencial:
    a. Cancelación de artefactos: componente que se encarga de eliminar los artefactos (ruido debido a otro tipo de actividad bioeléctrica como por ejemplo la que resulta del movimiento ocular o muscular) que contaminan la señal de entrada.
    b. Obtención de características: que traduce la señal cerebral de entrada en un vector de características en correlación con el fenómeno neurológico asociado a la señal.

   c. Traducción de características: que transforma el vector de características en una señal de control adecuada al dispositivo que se pretende controlar.
3. Aplicación: bloque funcional que recibe los comandos de control y realiza las acciones correspondientes en el dispositivo a través del controlador del mismo. En algunos sistemas BCI, la señal procesada es expandida o transformada a través del interfaz de control, por ejemplo, en el caso de un menú que permite diferentes acciones sobre el dispositivo (comandos) que son seleccionadas mediante el movimiento de un cursor (señal procesada).
4. Configuración: que permite a un operador definir los parámetros del sistema, como por ejemplo, determinadas variables para las diferentes etapas del procesado de señal. El operador no tiene por qué ser una persona técnica que ajuste el sistema BCI, sino que puede ser el propio usuario del sistema o, en el caso más deseable, algoritmos automáticos que ajustan el comportamiento del sistema en función de los resultados obtenidos y el feedback del usuario.

Dentro de las técnicas de posicionamiento de los electrodos EEG más utilizadas se encuentra la denominada 10-20,

El 10-20 System of Electrode Placement es un método que suele utilizarse para describir la localización de los electrodos en la cabeza. Estos electrodos son usados para grabar el electroencefalograma (EEG) usando una máquina llamada electroencefalógrafo. El EEG muestra la actividad cerebral y es el resultado de la actividad de millones de neuronas en el cerebro. El patrón de actividad cambia en función de la persona y del estado emocional de la misma.

El sistema 10-20 está basado en la relación entre la localización del electrodo y la parte del cerebro donde se encuentra colocada. Cada punto de la figura indica la posible posición del electrodo y cada sitio se identifica con una letra y un número u otra letra que identifica la localización del hemisferio. Las letras F, T, C, P y O representan Frontal, Temporal, Central, Parietal y Occipital, y los números pares (2, 4, 6,8) se refieren al hemisferio derecho, mientras que los impares (1, 3, 5,7) al izquierdo. La letra z se refiere a un electrodo colocado en el medio.

Como curiosidad decir que el "10" y el "20" se refieren al 10% o 20% de distancia entre los electrodos. En la ilustración 20 aparecen dos figuras con la posición vistas desde una perspectiva lateral y por encima de la cabeza.

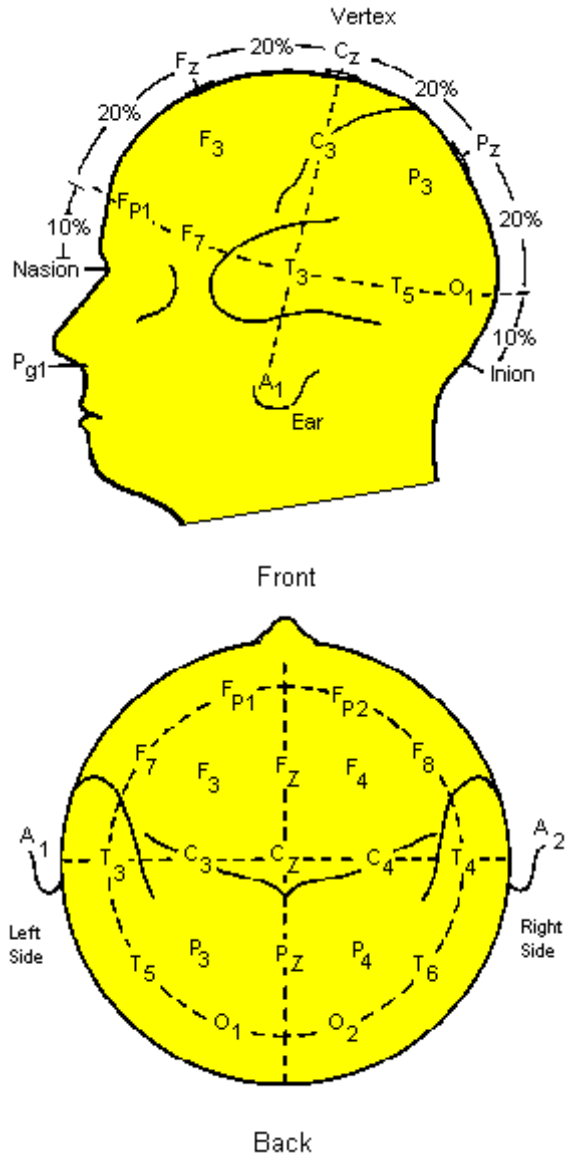


Ilustración 20. Sistema 10-20 EEG

Las características principales para determinar la calidad de un BCI son las siguientes:

1. Tiempo de entrenamiento requerido para el uso del sistema.

2. Tasas de transferencia.

3. Cantidad y tipo de errores en que se incurre comúnmente.

4. Satisfacción de los usuarios.

4.4. Trabajos realizados

Se han realizado numerosos trabajos sobre BCI, pero se ha intentado buscar los trabajos que pudieran ser más novedosos, y además que también estuvieran basados en técnicas evolutivas como la programación genética o los algoritmos genéticos. A continuación se explican dos trabajos que pueden resultar de interés para la realización de este proyecto:

4.4.1. Clasificación de señales de electroencefalograma usando programación genética

El primer trabajo [Alfaro05] trata de realizar la clasificación de señales de electroencefalograma usando para ello la programación genética. Se utilizaron dos técnicas de programación genética para intentar realizar la clasificación de la mejor manera posible.

Las funciones que se decidieron utilizar fueron las siguientes:

Función	Nº vars	Función	Nº vars
$\mathcal{R}$	0	logaritmo	1
exponencial	1	coseno	1
+	2	-	2
*	2	/	2
IfLTE	4		

Ilustración 21. Funciones utilizadas

Con este conjunto de funciones, que aunque se trata de funciones muy simples, se puede ver claramente como la conjunción de ellas puede dar lugar a árboles muy complejos, por lo que parece suficiente utilizar estas funciones. Con los datos de un individuo, se procede a realizar un procesado de los datos en serie de manera que se aplica el árbol adecuado y se obtiene una salida de manera que sirve para realizar la clasificación. La clasificación sólo se realiza sobre dos clases, es decir, sólo discriminará en dos clases distintas, de manera que si el valor de la función obtenida es menor que cero, clasificará a las clase C2 y si es mayor o igual que cero clasificará en la clase C1.

Todos los datos que se utilizan para realizar el trabajo han sido recogidos del concurso de BCI, y se eligen los datos para realizar el entrenamiento y posteriormente la validación. La clasificación se afirma en el trabajo que se realiza sobre el pensamiento del usuario de mover el cursor arriba y abajo, pero personalmente pienso que debido que las señales EEG toman la información de la parte sensorimotriz del cerebro, es más lógico pensar que el usuario tratará de pensar en mover un brazo o una pierna, y después esos movimientos se transcriben en mover el cursor hacia arriba o abajo.

Para realizar el software del algoritmo de programación genética se utilizó la librería JEO (Java envolving object) de Java, de manera que se modificaron algunas

características como la selección, que se realizaba mediante la ruleta y se cambió al método de los torneos, para aumentar la presión selectiva.

Para la realización del primer algoritmo se han utilizado las siguientes características, teniendo en cuenta que se ha realizado distribuido en dos islas, de manera que la ejecución finaliza cuando se sobrepasa un tiempo preestablecido.

Probabilidad de cruce	0.8
Probabilidad de clonación	0.1
Probabilidad de mutación	0.1
Método de inicialización	Ramped half and half
Profundidad máxima inicial de un árbol	7
Operador de cruce	Cruce de ramas
Probabilidad de selección de un nodo interno para cruce	0.9
Operador de mutación	Mutación de ramas
Operador de selección	Selección por torneo
Tamaño del grupo de torneo	10
Profundidad máxima de un árbol	18
Tamaño de la población	200/500

Ilustración 22. Parámetros del algoritmo de programación genética

Aquí se puede observar los parámetros del algoritmo, en los que el tamaño de la población empezaron siendo 200 y se cambió a 500 para realizar los dos últimos experimentos, cuyos resultados aparecen a continuación:

Nº	Aciertos entreno (en %)	Aciertos test (en %)	Nº individuos evaluados
1	71.6	83.6	3200
2	70.9	70.3	6800
3	72.4	85.0	55200
4	73.1	79.2	11600
5	78.4	81.2	63200
6	72.8	85.7	6600
7	81.7	70.0	64000
8	79.5	82.9	35500
Media	75.1	79.7	
DvEst	4.1	6.3	

Ilustración 23. Primeros resultados algoritmo de programación genética

En la figura 23 se pueden ver los resultados tanto en entrenamiento, como en test, así como el número de individuos que se han evaluado para cada experimento. Como se puede observar se ofrecen unos resultados bastantes buenos que como se dice en el trabajo se encuadrarían en la cuarta posición del campeonato de BCI, y también se

observa como los resultados de validación mejoran notablemente a los resultados de entrenamiento, este fenómeno no ocurre sólo en este trabajo, sino que ya ha sido advertido en otros experimentos y la posible explicación consiste en que los datos de validación fueron recogidos después de los de entrenamiento y entonces el usuario ya había adquirido práctica en el uso del BCI.

Cabe destacar como el experimento 7, que es el que mejor resultado ofrece en el entrenamiento es después el que peor resultado ofrece en la validación, y esto es debido a la sobreadaptación.

Para mejorar el problema de la sobreadaptación se decidió realizar el segundo algoritmo, que consiste en utilizar una técnica de muestreo aleatorio sobre el algoritmo de programación genética. Esta técnica consiste en utilizar un subconjunto del conjunto de vectores de entrenamiento en el proceso de evaluación en vez del conjunto entero. En cada generación ese subconjunto se elige aleatoriamente y se usa para la evaluación de todos los individuos de la población. Este método, además de disminuir el efecto de sobreadaptación, reduce considerablemente los tiempos de ejecución. En este caso, el subconjunto consta de 100 de los 268 vectores de entrenamiento. Utilizando esta técnica se obtienen los resultados de la figura 24:

Nº	Aciertos entreno (en %)	Aciertos test (en %)
2	72.0	84.3
5	70.5	85.3
11	77.2	86.3
21	71.6	84.3
22	70.1	85.0
25	74.6	86.7
31	70.5	84.3
33	72.0	86.7
Promedio	70.9	78.8
DesvEst	3.5	5.6

Ilustración 24. Segundos resultados algoritmo de programación genética

Se puede observar como los resultados obtenidos son bastante buenos, ya que todos se encontrarían entre la 4ª y la 6ª posición en el campeonato, y además se puede decir también que los resultados son parecidos a los que se obtuvieron con el otro algoritmo, pero además se ha mejorado el tiempo en alcanzar esos resultados y se evita el problema de la sobreadaptación.

4.4.2. Evolutionary Design of a Brain-Computer Interface

En este segundo trabajo [Romero05] los autores utilizan un algoritmo genético, unido a un perceptron multicapa para intentar resolver la clasificación en dos clases con los mismos datos utilizados por el primer trabajo.

Con la utilización conjunta de un algoritmo genético, junto con un perceptron multicapa, los autores afirman que conseguirán evitar el problema de la sobreadaptación y compararan los resultados con los de la competición.

En la explicación del algoritmo aparece descrito como utilizan el algoritmo genético para entrenar al perceptron multicapa, y como se obtuvieron los mejores resultados (sobre todo en tiempo) con un perceptron de una sola neurona y en una sola capa, es decir, un perceptron simple. Para ello, separaron los conjuntos de entrenamiento y de test, y aplicaron el entrenamiento sin necesidad de poseer ningún conocimiento sobre el significado de cada uno de los electrodos.

En el algoritmo genético se utilizaron tres operadores, dos tipos de mutación y uno de selección. El primer operador de mutación se realizó siguiendo una distribución Gaussiana con centro en 0 y desviación típica de 1, mientras que para el segundo se utilizó una distribución normal entre -2 y 2. Ambos operadores se aplicaron sobre un gen de un individuo en cada generación.

En lo referente a la selección de los parámetros adecuados, se dieron cuenta que utilizando poblaciones mayores de 200 individuos, el coste computacional aumentaba considerablemente y no se obtenían mejoras sustanciales con cada generación. En lo referente al número de generaciones a utilizar, puede parecer que elegir un número de generaciones muy grande, puede llegar a producir efectos de sobreadaptación, pero se puede observar como al llegar a un cierto número de generaciones se mantiene constante y no se produce este efecto.

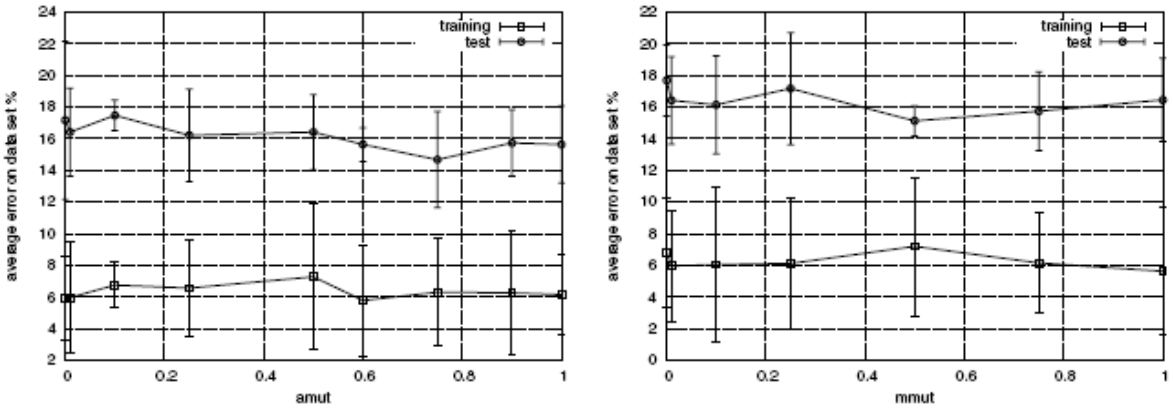


Ilustración 25. Comparación de resultados con el algoritmo de programación genética

Los mejores resultados que se han obtenido fueron con un diseño del perceptron de un perceptron simple y con un perceptron multicapa con dos neuronas ocultas y una neurona de salida.

Comparando los resultados con los de la competición, se afirma que no son mejores que los del primero en dicha competición, pero sin embargo son mejores que los que se han obtenido utilizando esta técnica actualmente. Los autores afirman que utilizar esta técnica resulta adecuado para este tipo de problemas, y se podría utilizar esta técnica para tratarlo como un problema multiobjetivo de fitness compartido.

5. Diseño general del sistema para la evolución de FLN con un enfoque multiobjetivo

En el presente capítulo se explica la idea desarrollada en el proyecto sobre la evolución de functional link networks con un enfoque multiobjetivo y el diseño realizado para poder llevarla a cabo. En ella, se deben tener en cuenta características importantes como la función de fitness, la codificación del cromosoma, el algoritmo utilizado y la obtención del resultado final.

5.1. Introducción

FLN utilizan combinaciones no lineales de los atributos originales y que sirven como entrada al clasificador. Estas combinaciones no lineales, que en el caso particular de este proyecto serán los productos entre los atributos originales, pueden ser un inconveniente debido a que podrían dar lugar a un número elevado de entradas, en los casos en los que el dominio cuente con un gran número de atributos de entrada. Para abordar este problema se propone realizar una evolución sobre FLN.

En [Sierra01] se propone la evolución de FLN, aplicando un algoritmo genético que permite realizar la evolución de manera que se puedan elegir como entradas a la red los atributos no lineales más influyentes a la hora de resolver el problema. Un EFLN evoluciona una población de FLN's que viene determinado por una codificación binaria del mismo que especifica los atributos del FLN dado que no existen capas ocultas en la red. La longitud del cromosoma binario viene determinada por el grado del polinomio que se elija y por la cantidad de atributos originales que tenga el dominio, que contabilizarán el número de términos polinomiales disponibles.

La principal dificultad puede consistir en elegir un adecuado grado polinomial para cada tipo de problema. De hecho, en algunos dominios complejos el utilizar simplemente un grado polinomial de dos, ya haría imposible el trabajar con todos los atributos producto y podría requerir un alto coste computacional. Para poder solucionar este problema, el algoritmo definido por Sierra [Sierra01] puede ser una buena solución. El algoritmo propuesto consiste en:

1. Ejecutar un AG con una población compuesta sólo de individuos de primer orden. (los atributos originales).
2. Guardar el fitness obtenido durante la ejecución anterior.
3. Repetir mientras no se encuentre una solución con un error de validación satisfactorio.
    a. Incrementar el grado polinomial (productos de atributos de grado n).
    b. Ejecutar el AG con una población de los mejores individuos encontrados al finalizar el anterior AG.
4. Devolver el mejor individuo o los mejores encontrados.

En el presente proyecto la idea consiste en plantear y validar para la evolución de FLN un enfoque multiobjetivo. A continuación se explican los objetivos planteados, la codificación del cromosoma, el esquema general del algoritmo y como se elegirá el punto del frente final para obtener un resultado final.

5.2. Definición de múltiples objetivos

Cuando se utiliza un algoritmo genético, una de las características más importantes consiste en la elección de la función de evaluación que se utilizará en la ejecución. Al tratarse de un algoritmo genético multiobjetivo, será importante, por tanto, definir cuáles serán estos objetivos. Se desarrollaron dos casos distintos en el que se variaron los objetivos con el fin de conocer los resultados que podrían obtener.

Para el primer caso se definieron dos objetivos: minimizar el número de fallos obtenido por el clasificador y minimizar el número de atributos usados por el clasificador. Este último objetivo se ha introducido con la finalidad de obtener un modelo que posea buenas propiedades de generalización, ya que en términos generales, al minimizar el número de atributos del modelo, mejores propiedades de generalización se esperan, y por tanto, se espera conseguir un mejor resultado con el porcentaje de test.

En el segundo caso se intentó dar un enfoque de manera que se pudiera analizar el comportamiento del sistema cuando los objetivos son minimizar el porcentaje de fallos para cada una de las clases del dominio a clasificar, de manera que el objetivo principal consistente en minimizar el número de fallos total, fuera dividido en objetivos dedicados a minimizar el número de fallos para cada clase y con ello se obtuviera un frente con soluciones diversas. Con este caso se intenta estudiar cómo se distribuye el frente de soluciones no dominadas, para analizar si existen modelos capaces de equilibrar el porcentaje de aciertos por clases.

5.3. Codificación del cromosoma

A la hora de codificar el cromosoma es importante tener en cuenta el grado polinomial. En este caso y basándonos en los resultados de Sierra [Sierra01] se ha decidido utilizar polinomios de grado dos, ya que se mostró que utilizar polinomios de grado mayor no ayudaba a mejorar la clasificación.

Se realizará una codificación binaria de manera que indique si un atributo del individuo, ya sea original o producto se utilizará como entrada a FLN para realizar la clasificación. El individuo tendrá la forma que se puede apreciar en la figura 26:

X_1	X_2	…	X_n	$X_1 X_2$	$X_1 X_3$	…	$X_{n-1} X_n$

Ilustración 26. Individuo

El individuo por tanto estará formado por los atributos originales $\{x_1, x_2,\ldots, x_n\}$ y los productos de los atributos originales. Dado n atributos originales, la longitud del cromosoma será

$$n + \sum_{i=1}^{n-1} i = n + n(n-1)/2 = n(n+1)/2$$

5.4. Esquema general del algoritmo

En la figura 27 se muestra un diagrama de bloques para comprender más fácilmente el sistema desarrollado.

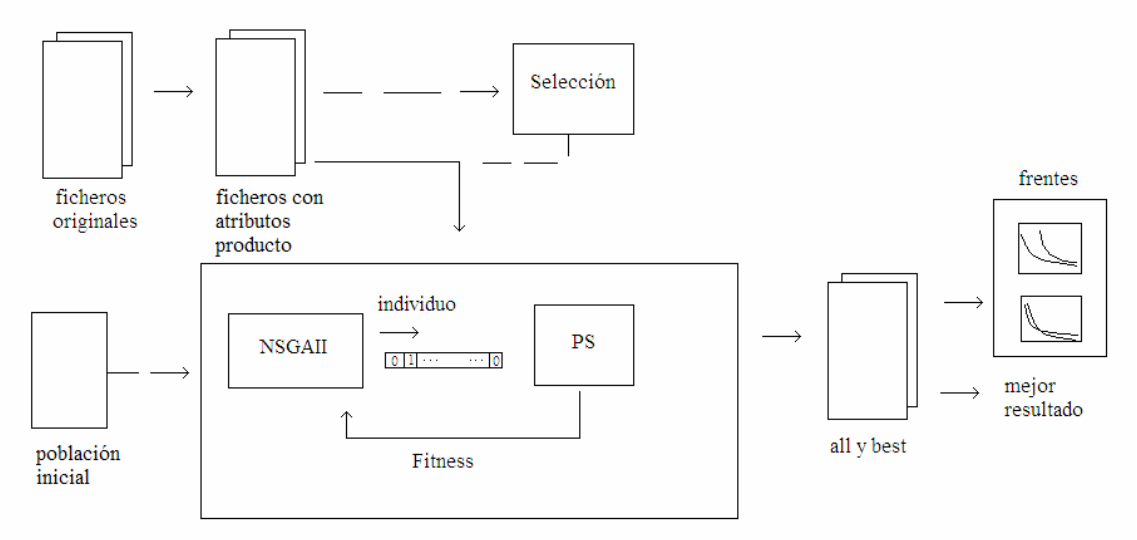


Ilustración 27. Diagrama bloques del sistema desarrollado

Partiendo de los ficheros originales del dominio a tratar, será necesario obtener un fichero en el que se incluyan los atributos producto para los conjuntos de entrenamiento, test y validación. En algunos casos cuando el dominio tiene muchos atributos originales, la inclusión de una cantidad enorme de atributos producto puede ralentizar la ejecución del código y hacer que las pruebas se eternicen o incluso lleguen a utilizar una cantidad tal de memoria que las pruebas no puedan realizarse. En estos casos es necesario realizar una selección sobre los atributos producto. También se puede opcionalmente inicializar el sistema con una población inicial que se puede definir previamente.

Una vez generados los atributos producto, se procede a ejecutar el algoritmo genético multiobjetivo, que será el algoritmo NSGAII, que ya se explicó en detalle en el apartado 3.4 "NSGAII", y cuyo código es de libre distribución y se encuentra en http://www.iitk.ac.in/kangal/codes.shtml . Fue realizado por K. Deb y sus alumnos del Laboratorio de Algoritmos Genéticos del Instituto Tecnológico Kanpur en India [Deb00]. A partir de una población inicial, actúa el algoritmo NSGAII utilizando los operadores de mutación y cruce y realizando ordenaciones no dominadas de población y utilizando técnicas como la distancia de crowding para obtener poblaciones diversas.

Para cada individuo de la población del algoritmo NSGAII es necesario evaluar su fitness. Para dicha evaluación, en primer lugar se construye el modelo de FNL utilizando los atributos productos que estén representados por un 1 en el cromosoma. A continuación y utilizando algunas de las implementaciones del perceptron explicadas en el apartado 2 "Introducción de no linealidades en el perceptron simple", se entrena el perceptron correspondiendo con el conjunto de datos de entrenamiento y se obtienen los diferentes objetivos dependiendo del caso utilizado.

La elección de una implementación u otra del perceptron en este proyecto ha venido dada por el número de clases del dominio. Para los dominios con dos clases, se ha utilizado un perceptron simple con razón de aprendizaje, ya explicado en el apartado 2.1 "Perceptron simple". Y para el dominio con tres clases se han utilizado las implementaciones de dos y tres perceptrones simples combinados ya explicados en el apartado 2.2 "Diferentes implementaciones del perceptron simple para problemas de clasificación".

Cuando todas las generaciones hayan finalizado y se escriban los ficheros de salida, el código de NSGAII volverá a llamar a la función de evaluación con cada uno de los individuos que componen el frente final. Será necesario volver a entrenar a estos individuos con un número de ciclos de aprendizaje suficiente para garantizar la convergencia del clasificador, ya que no se guardan los valores de los pesos de una ejecución de los perceptrones a otra. Se ha realizado así pues se puede considerar que guardar los valores de los pesos y umbrales para cada uno de los individuos y luego buscar el individuo y realizar la correspondencia con sus pesos, podría ser incluso más costoso que volver a realizar este entrenamiento. Una vez entrenados los diferentes modelos que representan el frente, se evalúan dichos modelos para los conjuntos de test y validación.

En el algoritmo de aprendizaje de los perceptrones, los pesos y umbrales teóricamente se deben inicializar aleatoriamente, con lo que podemos pensar que dos ejecuciones distintas con el mismo individuo podrían devolver dos resultados distintos. Pero este problema se ha solucionado inicializando los valores de los pesos y el umbral de ambos perceptrones siempre a cero, con lo que siempre dos ejecuciones distintas de un mismo individuo deberían devolver el mismo resultado.

5.5. Elección del punto del frente

Al final de toda la ejecución de nuestro sistema, se obtiene un frente de soluciones no dominadas, donde cada punto del frente representa un modelo FLN. En el contexto de aprendizaje automático es necesario elegir el mejor punto del frente que representará el modelo obtenido. El mejor modelo no tiene que corresponder con el mejor resultado en términos de porcentaje de aciertos sobre el conjunto de entrenamiento, pues dicho modelo podría generalizar mal.

Para elegir entonces el mejor modelo, se utiliza un conjunto de test. Se evalúa cada punto del frente sobre dicho conjunto y se toma el punto correspondiente con el mejor porcentaje de acierto en clasificación. Finalmente, y sólo con el objetivo de mostrar la capacidad de generalización del modelo, se obtiene el porcentaje de aciertos sobre un tercer conjunto de validación.

5.6. Algunas consideraciones adicionales

Después de realizar algunas pruebas y tras darnos cuenta de que en el primer caso, en el que los objetivos son minimizar el número de atributos y el porcentaje total de fallos, observamos viendo los frentes que se obtenían como resultado, que siempre el primer frente partía de individuos con aproximadamente un 50% de 1`s en su cromosoma y a partir de ahí iba disminuyendo. Con este tipo de pruebas el espacio de búsqueda quedaba reducido a individuos con un 50% de 1's hacia abajo, quedando

olvidados en los frentes todos los individuos que contuvieran un porcentaje de 1's mayor. Para tratar de evitar este problema se pensó en forzar la inicialización de los individuos de manera que este porcentaje de 1's fuera mayor y ampliar así el espacio de búsqueda. De esta manera el espacio de búsqueda sobre el que actuaba el algoritmo es mayor, aunque para encontrar soluciones buenas con pocos 1's, es decir, que utilizaran pocos atributos en el cromosoma, fuera necesario realizar alguna generación más del algoritmo. Para ello, la inicialización de los individuos se ha realizado de manera que el porcentaje de 1's sea del 80%. Los frentes obtenidos en las diferentes simulaciones poseen un número de atributos productos inferior al dicho porcentaje.

Al utilizar un algoritmo evolutivo, en cada generación y para cada individuo es necesario entrenar un perceptron simple, lo cual puede implicar un elevado tiempo en la evaluación de los individuos. Una forma de graduar el número de iteraciones que ejecutan los perceptrones para cada generación del algoritmo NSGAII consiste en ir aumentando linealmente el número de iteraciones del perceptron en función del número de generaciones del algoritmo, de manera que el número de iteraciones se establece para el o los perceptrones simples en la generación x es:

$Iteraciones_x = \lfloor Nx / 2M \rfloor + N / 2$ siendo:

N el número máximo de ciclos de aprendizaje.
M el número de generaciones máximo del algoritmo NSGAII.
x el número de generación en la que se encuentra el algoritmo.

Por tanto, el número de iteraciones comenzará con la mitad de las iteraciones definidas como máximo en la primera generación e irá aumentando linealmente hasta que en la última se ejecutan todas las iteraciones.

6. EXPERIMENTACIÓN

Este capítulo trataremos toda la experimentación realizada sobre los diferentes dominios analizados, en primer lugar el BCI. A continuación se tratan el cáncer, la diabetes y un dominio artificial denominado cuadrático. Se realizará primero una breve descripción de los datos, para después exponer los resultados con los perceptrones simples y después con el software realizado.

6.1. Descripción de los datos BCI

Los datos del BCI han sido obtenidos de la competición anual sobre Brain Computer Interfaces, para ser más concretos se tratan de los de la competición de 2005, y se pueden obtener de la siguiente página web http://ida.first.fraunhofer.de/projects/bci/competition_iii/. Este conjunto de datos se refiere a tres sujetos durante cuatro sesiones independientes. Los sujetos se encontraban sentados sobre una silla y con los brazos relajados sobre sus piernas y no tenían ninguna experiencia previa sobre la realización de este tipo de experimentos. A los sujetos se les pedía que realizaran tres tareas:

1. Imaginar movimientos repetitivos de la mano izquierda. (left, clase 2).
2. Imaginar movimientos repetitivos de la mano derecha. (right, clase 3).
3. Generar palabras comenzando por una letra al azar. (word, clase 7).

Las cuatro sesiones de cada sujeto se hicieron en el mismo día, en tiempos de cuatro minutos y con descansos de 5-10 minutos entre ellos [Mill04]. El sujeto realizó una tarea dada durante aproximadamente quince segundos y luego cambió al azar a otra tarea mental.

Los datos son proporcionados de maneras distintas:

1. Señales EEG, con un muestreo de 512 Hz.
2. Datos precalculados, en el que las señales EEG han sido preprocesadas.

Cada muestra EEG en los datos preprocesados contiene un vector de dimensión 96, ya que contiene ocho canales con doce componentes de frecuencia en cada uno de ellos. Para cada sujeto hay tres ficheros de entrenamiento y un fichero de test, que se encuentran en formato ASCII y que contienen los siguientes ejemplos en cada caso.

	ENTRENAMIENTO	TEST
SUJETO 1	3488/3472/3568	3504
SUJETO 2	3472/3456/3472	3472
SUJETO 3	3424/3424/3440	3488

Tabla 2. Número de ejemplos en cada fichero para cada sujeto

Las señales EEG fueron registradas con un sistema Biosemi que usa una especie de gorro con 32 electrodos integrados localizados en las posiciones estándar del sistema 10-20. La frecuencia de muestreo era 512 Hz y las señales fueron adquiridas con corriente continua y sin utilizar ningún tipo de corrección.

La separación de los ficheros viene dada por el mismo dominio, de manera que se realizaron cuatro sesiones. Las dos primeras serán para entrenamiento, la tercera para test y la cuarta para validación.

Se realizó un estudio sobre los datos que se encuentra en el ANEXO 1, y en el que se realiza una exploración sobre los mismos, aplicando distintos algoritmos de aprendizaje automático sobre los conjuntos de datos de los tres sujetos y obteniéndose diversos resultados, de los cuales el mejor se comparó con los de la competición BCI. En nuestro caso, se han borrado algunas partes del ANEXO 1, que no eran de interés ya que para nuestras pruebas posteriores sólo se utilizarán los datos referidos al sujeto 1.

Ya que sólo trabajaremos sobre el sujeto 1, podemos ver como en la competición los resultados sobre este sujeto superan ligeramente el 70% de aciertos en la mayoría de los casos, alcanzándose el 80% en el caso del ganador de la competición por tanto, este será el resultado al que nos deberemos de aproximar para comprobar que la técnica utilizada en este proyecto es capaz de ofrecer buenos resultados. (http://ida.first.fraunhofer.de/projects/bci/competition_iii/results/index.html)

6.2. Resultados BCI con Perceptrones Simples

Con el dominio de BCI se realizaron varias implementaciones de clasificadores basados en perceptrones simples, de manera que se llevaron a cabo pruebas variando el clasificador, además de variar sus parámetros, como son las iteraciones y en algunos casos la razón de aprendizaje. Aparte de variar las características propias de los clasificadores, fue necesario también realizar las pruebas sobre los 96 atributos originales y sobre los que se escogieron después de la última selección (387 como se explica posteriormente en el apartado 6.3 "Resultados BCI con el software realizado"). Para todas estas pruebas aparecen reflejados los resultados para los conjuntos de entrenamiento, test y validación así como el tiempo que la prueba tardaba en realizarse. Se escogió como conjunto de entrenamiento los ficheros con los datos desordenados.

Con estas pruebas trataremos de encontrar la mejor implementación de perceptrones con el fin de obtener, además de éste perceptron, los parámetros aproximados con los que probar las posteriores pruebas. A la hora de medir los resultados, no sólo tendremos en cuenta el resultado obtenido, sino también el tiempo.

Para todas las pruebas que se han realizado se mostrarán los resultados para los conjuntos de entrenamiento, test y validación en porcentaje de aciertos y en segundos el tiempo de ejecución.

6.2.1. Pruebas 96 atributos

Para realizar estas pruebas se cogieron los ficheros con los atributos originales, además para el fichero de entrenamiento se realizó una desordenación de los datos para evitar los problemas que se explicaron en el apartado 6.3.3 "Pruebas con 387 atributos y datos desordenados" y los ficheros de test y validación se mantuvieron iguales.

6.2.1.1.Prueba 2 PS con 96 atributos

Se realizaron pruebas con diferentes combinaciones de iteraciones para ejecutar los dos perceptrones y además el valor de las repeticiones se eliminó con el fin de que se ejecutarán todas las iteraciones que se deseaban y además de observar si se podía obtener un mejor resultado, medir también el tiempo que se tardaba en realizar cada una de las pruebas, en la tabla 3 se muestran los resultados.

ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
50	**65.85**	**67.93**	**63.49**	**4.6**
100	**67.97**	**66.78**	**63.35**	**6.9**
200	**71.62**	**72.05**	**65.32**	**12.1**
300	**71.62**	**72.05**	**65.32**	**15.5**

Tabla 3. BCI: Resultados 2PS + 96 Atributos

El 65.32% de aciertos en validación será el resultado que deberemos mejorar al integrar este implementación de perceptrones en nuestro software, ya que si utilizamos FLN, lo mínimo esperable es que se superen los resultados que si no se utilizase esta opción.

6.2.1.2.Prueba 3 PS con 96 atributos

En este caso aparte de variar las iteraciones de los perceptrones, también será necesario variar la razón de aprendizaje, modo que se obtengan resultados para estas combinaciones. Se utilizó la razón de aprendizaje de 0.1 y de 0.8 con todas las combinaciones de iteraciones y para las demás razones de aprendizaje sólo se probó con 300 iteraciones. En la tabla 4 se pueden ver los resultados.

RAZÓN DE APRENDIZAJE	ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
0.1	**50**	**59.62**	**65.24**	**64.89**	**5.2**
0.1	**100**	**59.62**	**65.24**	**64.89**	**9.1**
0.1	**200**	**69.84**	**68.46**	**67.63**	**17**
0.1	**300**	**75.11**	**74.85**	**71.54**	**24**
0.5	**300**	**73.42**	**66.5**	**67.09**	**24.6**
0.8	**50**	**76.91**	**75.22**	**73.2**	**5.7**
0.8	**100**	**77.35**	**75.08**	**72.63**	**9.1**
0.8	**200**	**77.48**	**75.14**	**72.54**	**17**
0.8	**300**	**77.48**	**75.14**	**72.54**	**24.5**
1	**300**	**77.84**	**75.53**	**72.3**	**24.6**

Tabla 4. BCI: Resultado 3PS + 96 atributos

Como se puede ver los mejores resultados se obtienen con una razón de aprendizaje de 0.8, y además los resultados obtenidos son bastante buenos, siendo curioso que el mejor resultado para validación se haya conseguido con sólo 50 iteraciones para el perceptron, mientras que si se aumentan estas generaciones el resultado para validación empeora ligeramente. El mejor resultado obtenido tiene un 73.2% de aciertos en validación, y sería adecuado mejorarle utilizando la integración de tres perceptrones simples en el software realizado.

6.2.1.3.Prueba 1 PS con 96 atributos

Para esta prueba igual que en la anterior, se variarán las iteraciones de los perceptrones, así como la razón de aprendizaje. En esta prueba queremos comprobar si en comparación con las dos anteriores, el tiempo de ejecución es bastante menor, siendo los resultados no inferiores a las anteriores pruebas. Los resultados se pueden ver en la tabla 5.

RAZÓN DE APRENDIZAJE	ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
0.1	**50**	**47,45**	**56,47**	**53,02**	**3,9**
0.1	**100**	**52,16**	**60,36**	**56,67**	**6,2**
0.1	**200**	**55,48**	**61,85**	**59,04**	**10,5**
0.1	**300**	**56,83**	**62,13**	**60,04**	**15**
0,1	**1000**	**59,6**	**62,3**	**61,21**	**40,5**
0.5	**300**	**62,06**	**62,61**	**61,9**	**11,5**
0.8	**300**	**62,93**	**63,28**	**62,55**	**12,6**
1	**300**	**63,06**	**63,36**	**62,58**	**12,6**
1	**1000**	**64,32**	**63,45**	**63,15**	**35,9**

Tabla 5. BCI: Resultado 1PS + 96 atributos

En este caso podemos comprobar cómo a pesar que el tiempo de ejecución efectivamente es menor que en las pruebas anteriores, sin embargo, los resultados son bastante peores.

6.2.2. Pruebas 387 atributos

Los 387 atributos se obtienen de una selección realizada como se explica posteriormente en el apartado 6.3 "Resultados BCI con el software realizado". Para realizar estas pruebas se cogieron los mismo ficheros que con los que se harán las pruebas sobre todo el software que se ha desarrollado, además se utilizará el fichero de entrenamiento que tiene los datos desordenados para poder ofrecer un resultado más fiable sobre cuál será o serán los clasificadores que se integren sobre nuestro software.

6.2.2.1.Prueba 2 PS con 387 atributos

Se realizaron pruebas con diferentes combinaciones de iteraciones para ejecutar los dos perceptrones y además el valor de las repeticiones se eliminó con el fin de que se ejecutarán todas las iteraciones que se deseaban y además de observar si se podía obtener un mejor resultado, medir también el tiempo que se tardaba en realizar cada una de las pruebas, en la tabla 6 se muestran los resultados de estas pruebas.

ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
20	**58.17**	**65.02**	**59.21**	**8.6**
50	**70.25**	**69.93**	**66.35**	**16**
100	**70.25**	**69.93**	**66.35**	**23**
200	**70.25**	**69.93**	**66.35**	**43**
300	**70.25**	**69.93**	**66.35**	**60**

Tabla 6. BCI: Resultado 2PS + 387 atributos

Las pruebas muestran como con 50 iteraciones se llegó al mejor resultado posible, y como a pesar de que se aumenten las iteraciones el resultado no mejora. Se puede observar además como el resultado en entrenamiento mejora el de test y sobre todo el de validación, como el tiempo aumenta notablemente al aumentar las iteraciones hasta llegar a un minuto sobre las 300 iteraciones. El 66.35% de validación será el resultado a mejorar por el software a realizar integrando esta implementación de perceptrones.

6.2.2.2.Prueba 3 PS con 387 atributos

Para realizar estas pruebas será necesario variar el número de iteraciones, así como la razón de aprendizaje, teniendo en cuenta el resultado que se obtenga, así como el tiempo que se obtenga.

RAZÓN DE APRENDIZAJE	ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
0.1	**50**	**36.42**	**28.47**	**30.02**	**19**
0.1	**100**	**37.02**	**29.96**	**31.33**	**36**
0.1	**200**	**64.71**	**65.7**	**66.09**	**69**
0.1	**300**	**64.71**	**65.7**	**66.09**	**97**
0.8	**300**	**60.1**	**62.13**	**58.53**	**155**
1	**300**	**66.75**	**68.97**	**67.38**	**152**
0.5	**20**	**45.4**	**47.56**	**46.6**	**15.6**
0.5	**50**	**73.6**	**72.16**	**71.17**	**22**
0.5	**300**	**74.72**	**72.42**	**71.51**	**152**
0.5	**1000**	**74.74**	**73.87**	**72.29**	**312**

Tabla 7. BCI: Resultado 3PS + 387 atributos

Los resultados obtenidos son bastante buenos, siendo el mejor resultado con la razón de aprendizaje de 0.5 y siendo el mejor resultado un 72.29% para validación, pero para obtener este resultado se necesitaron 1000 ciclos de aprendizaje para los perceptrones, lo que supuso un tiempo de ejecución superior a los cinco minutos. Los resultados son buenos y por tanto, será interesante integrar esta combinación de perceptrones en nuestro software, pero habrá que tener en cuenta las iteraciones que se realizan sobre las pruebas, ya que las pruebas podrían tardar demasiado tiempo en ejecutarse, viendo también que los resultados con 300 iteraciones tardan unos 2.5 minutos, podrían ser también demasiado.

6.2.2.3.Prueba 1 PS con 387 atributos

En este caso, realizaremos pruebas variando la razón de aprendizaje y las iteraciones del perceptron, pero estas combinaciones variarán en función de los resultados obtenidos, de manera que se intentará llegar a los resultados de las pruebas con otros perceptrones, variando las iteraciones de manera que el tiempo sea parecido.

RAZÓN DE APRENDIZAJE	ITERACIONES	TRAIN	TEST	VALIDACIÓN	TIEMPO
0.1	**50**	**46.92**	**57.06**	**53.53**	**15.4**
0.1	**100**	**51.3**	**60.2**	**56.9**	**22.4**
0.1	**200**	**55.81**	**62.19**	**59.67**	**35.8**
0.1	**300**	**57.19**	**62.3**	**59.93**	**50**
0,1	**1000**	**60.9**	**63.7**	**62**	**143**
0,1	**2000**	**62.16**	**64.79**	**63.09**	**260**
0.5	**300**	**57.09**	**60.67**	**60.33**	**50**
0,5	**1000**	**59.75**	**62.38**	**61.21**	**143**
0,8	**1000**	**59.03**	**61.4**	**60.7**	**143**
1	**1000**	**58.9**	**61.09**	**60.73**	**140**

Tabla 8. BCI: Resultado 1PS + 387 atributos

Después de comprobar que, aún aumentando el número de iteraciones de manera que el tiempo se asemeje al de las pruebas con los otros perceptrones, los resultados obtenidos no son competitivos con los otros dos, se decidirá descartar la integración de la implementación de este perceptron con el software realizado, ya que se asume que los resultados que se obtendrían no llegarían a los que se podrían obtener con las otras implementaciones.

6.3. Resultados BCI con el software realizado

Para el dominio de BCI tenemos 96 atributos originales, los atributos productos que se generarían, serían 4656. Con esta cantidad de atributos el coste computacional de ejecutar el algoritmo sería enorme y por ello se realizaron numerosas pruebas con el fin de obtener los atributos más relevantes. Realizando algunas pruebas con Weka se puede ver como quitando de cada canal los seis últimos componente de frecuencia, los resultados que se obtienen son prácticamente los mismos, y por tanto, se decidió que los atributos producto sólo se realizarían sobre estos 48 atributos. Partiendo de 48 atributos originales los atributos productos que se generarían, serían 1176.

Se realizaron algunas pruebas con este número de atributos, después se decidió realizar una selección debido al alto coste computacional que todavía existía. Para realizar esta selección se utilizó la herramienta Weka y en la que se cogió el fichero de entrenamiento y se realizaron unas cuantas pruebas sobre la selección de atributos. Se utilizó el algoritmo Ranker que proporciona una ordenación según la correlación entre los datos y la clase, se observó que los 339 primeros atributos producto (de entre los 1176) eran los más relevantes, por lo que se seleccionaron éstos, en adición a los 48 atributos originales, resultando en 387 atributos.

Inicialmente se realizó una seria de experimentos que proporcionaban unos resultados bastante malos (alrededor de un 50% de aciertos con tres clases). Posteriormente se descubrió que los datos estaban ordenados por clase, lo que causaba que los perceptrones simples no aprendieran bien. Este hecho hizo que los experimentos tuvieran que ser descartados, aunque sirvieron para comprobar que el software desarrollado funciona correctamente, tanto para el caso 1 como para el caso 2, y que la utilización de conjuntos de datos con 387 atributos es manejable por el sistema (no así el utilizar los 1176 atributos, que exige un tiempo excesivo). Por tanto, a continuación todos los experimentos utilizan un conjunto de entrenamiento con los datos desordenados aleatoriamente.

6.3.1. Pruebas con 387 atributos con dos perceptrones simples

En este apartado se describen varios experimentos del sistema realizado, para los casos uno y dos, utilizando los datos con los 387 atributos (atributos originales y atributos producto), estando los datos para el conjunto de entrenamiento desordenados aleatoriamente.

6.3.1.1.Prueba 1

Se realizó una prueba muy sencilla con el fin de comprobar el funcionamiento del sistema. Para ello se escogió el caso 1, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total, con una población de 32 individuos, 4 generaciones y 300 iteraciones para el perceptron. No es necesario ver la evolución del frente, ya que se trata sólo de 4 generaciones. El mejor resultado en test de aproximadamente un 28.1% y en validación de un 28.5%.

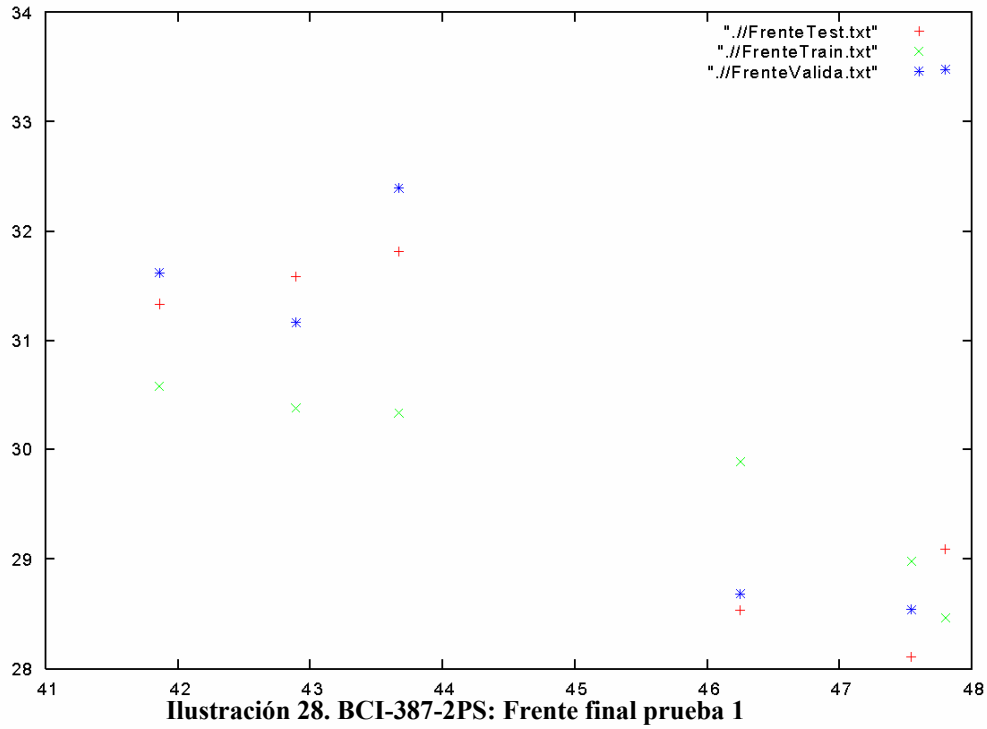


Ilustración 28. BCI-387-2PS: Frente final prueba 1

6.3.1.2.Prueba 2

Para esta prueba se aumentaron los individuos y generaciones para tratar de encontrar mejores resultados. Con 100 individuos y 100 generaciones sobre 300 iteraciones del perceptron. En la figura 29 se muestra la evolución del frente.

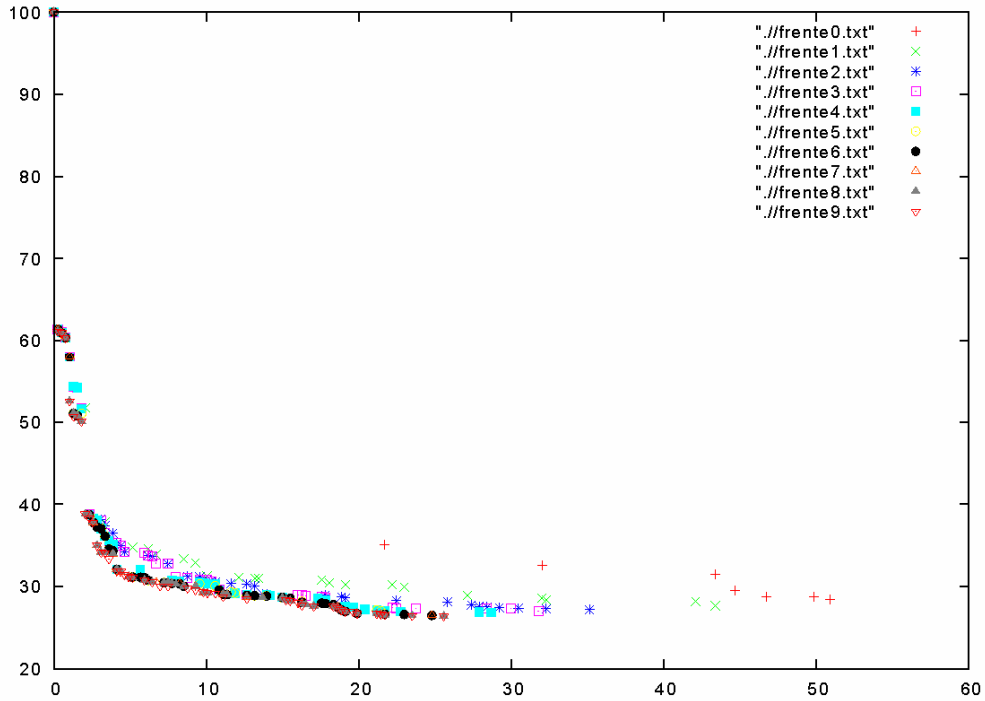


Ilustración 29. BCI-387-2PS: Evolución frente prueba 2

Después de ver cómo evoluciona el frente, en la figura 30 se muestra los frentes finales en las que el resultado es algo peor que la prueba anterior, ya que aunque para el conjunto de test el resultado es de un 26.4% para el conjunto de validación es de un 32.5%.

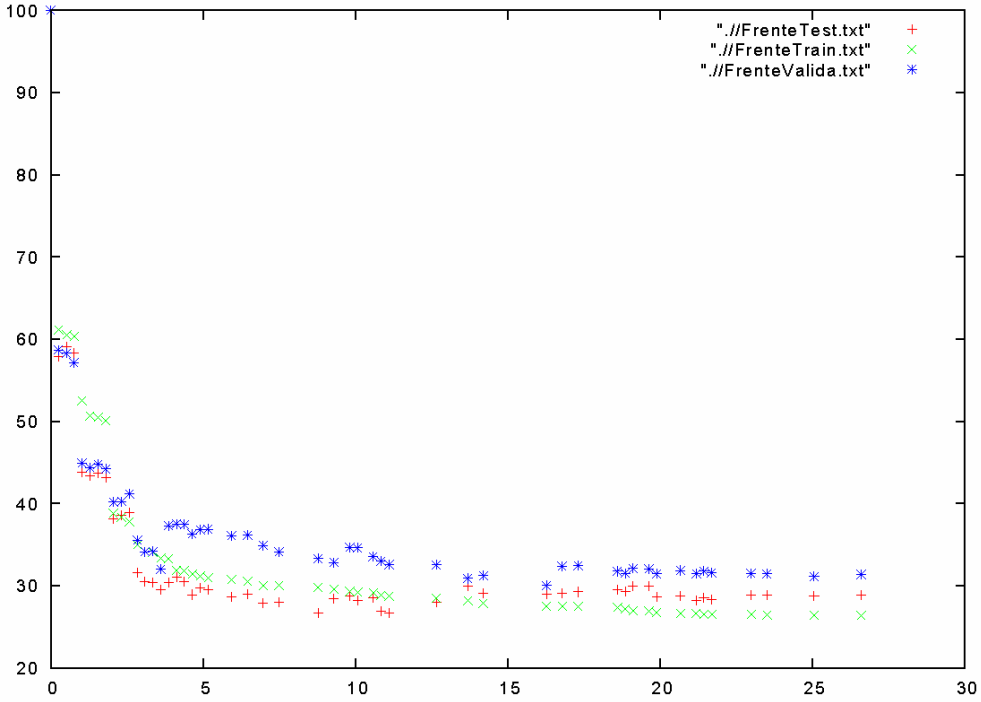


Ilustración 30. BCI-387-2PS: Evolución frente prueba 2

6.3.1.3.Prueba 3

Para la siguiente prueba se trataron de aumentar el número de generaciones para comprobar si se obtenía un mejor resultado. El número de individuos es de 48 con 200 generaciones y 300 iteraciones para el perceptron. En la figura 31 se puede ver la evolución de los frentes.

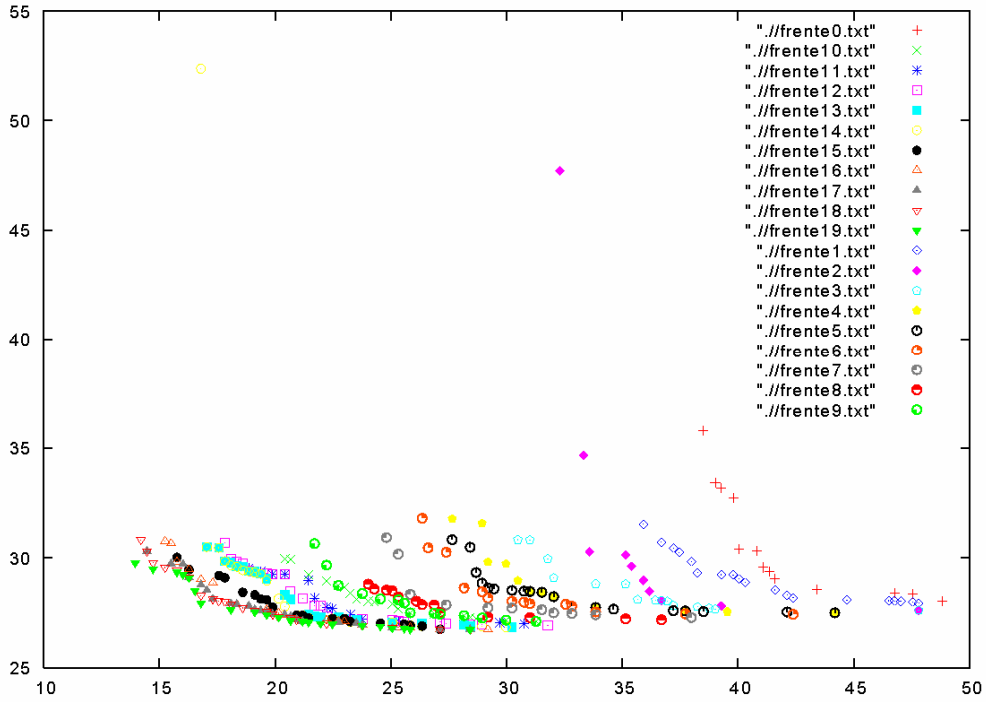


Ilustración 31. BCI-387-2PS: Evolución frente prueba 3

En esta prueba se observa una gran diferencia en los conjuntos de entrenamiento y test, con el de validación en algunos resultados, el mejor resultado para test es de un 26.9% y para validación de un 32.1% por lo que se puede decir que el resultado es semejante a la prueba anterior.

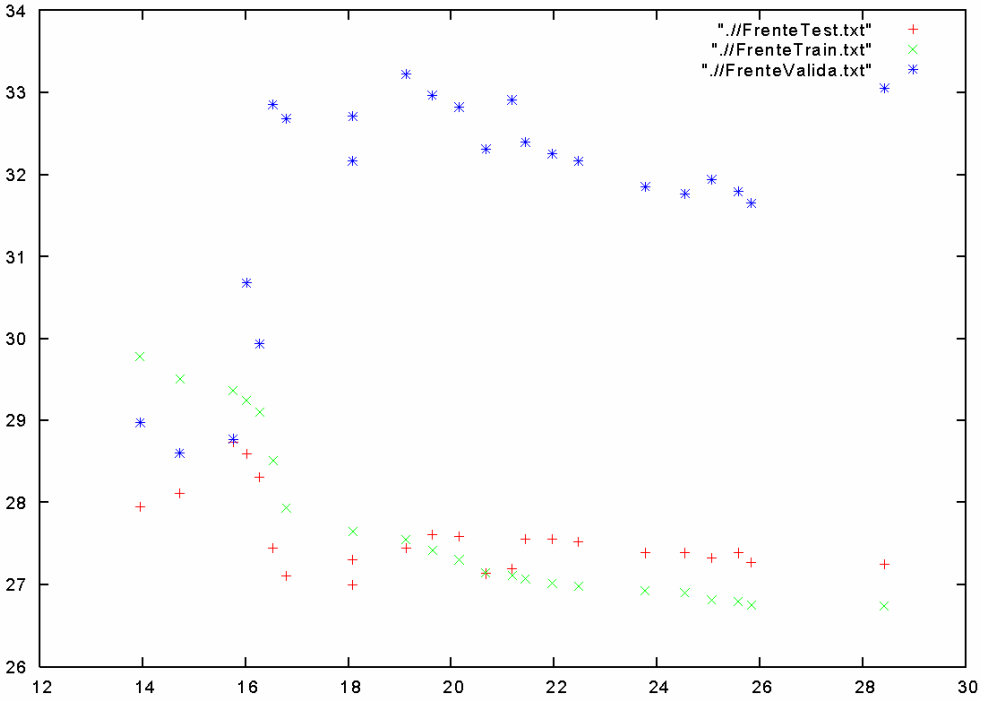


Ilustración 32. BCI-387-2PS: Frente final prueba 3

6.3.1.4.Prueba 4

Para la siguiente prueba se trató de disminuir el número de generaciones para comprobar si con este propósito se obtenía un mejor resultado. El número de individuos es de 100 con 30 generaciones y 300 iteraciones para el perceptron. En la figura 33 se puede ver la evolución de los frentes.

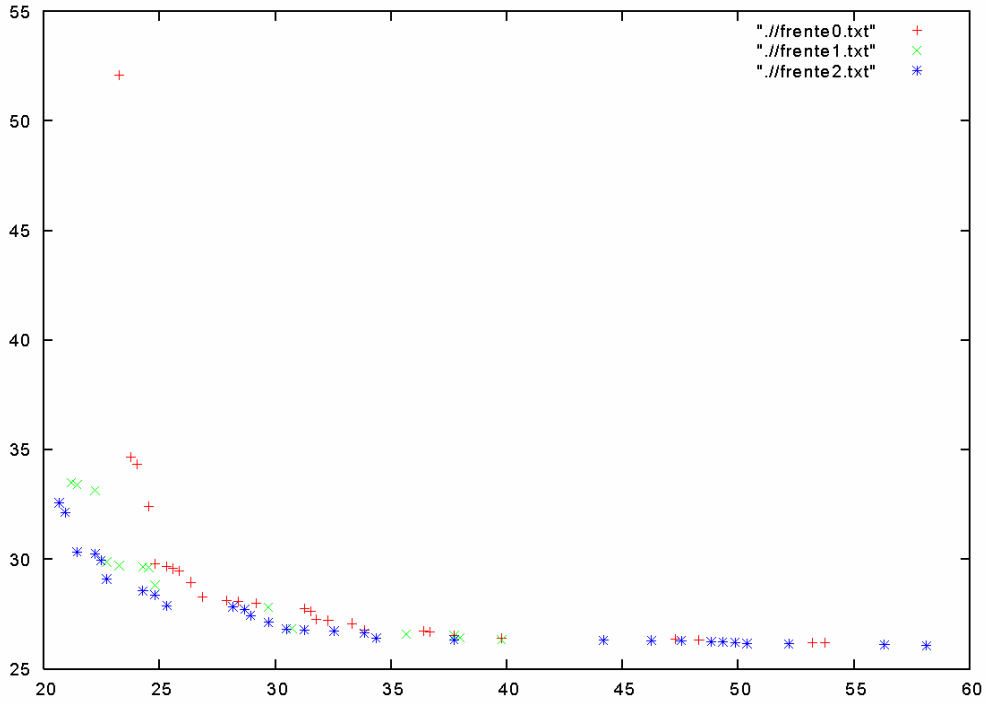


Ilustración 33. BCI-387-2PS: Evolución frentes prueba 4

En este caso los frentes que se muestran en la figura 34 muestran más puntos que en la prueba anterior, además de mostrar la misma diferencia entre validación y entrenamiento que la prueba anterior. El mejor resultado es de un 26.7% para test y de un 29.5% para validación. Como se puede ver los resultados varían pero en torno a un 3% de diferencia y además funciona mejor con pocas iteraciones.

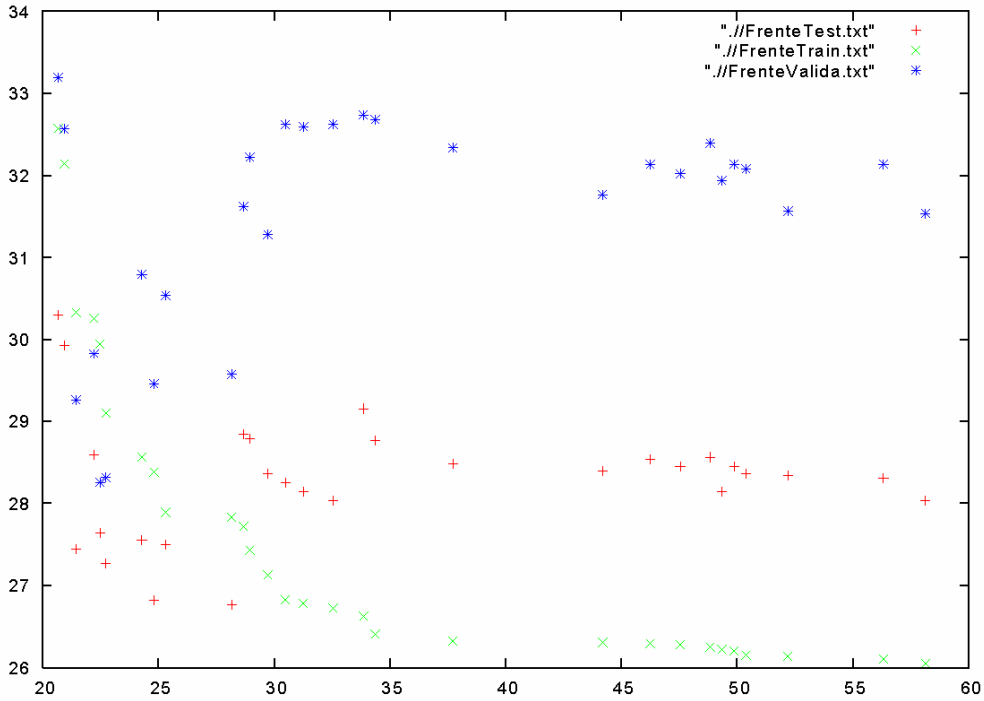


Ilustración 34. BCI-387-2PS: Frente final prueba 4

6.3.1.5.Prueba 5

Para esta prueba se aumentó el número de generaciones de manera que, serán 100 individuos y 100 generaciones sobre 300 iteraciones del perceptron. Además se trató de que el espacio de búsqueda fuera mayor con el fin de intentar unos resultados distintos, de manera que los individuos iniciales tuvieran una probabilidad de un 80% de tener 1's. Como se puede ver en la figura 35, lo individuos comienzan en el primer frente de la evolución con un porcentaje de un 80% de 1's y a partir de ahí van bajando.

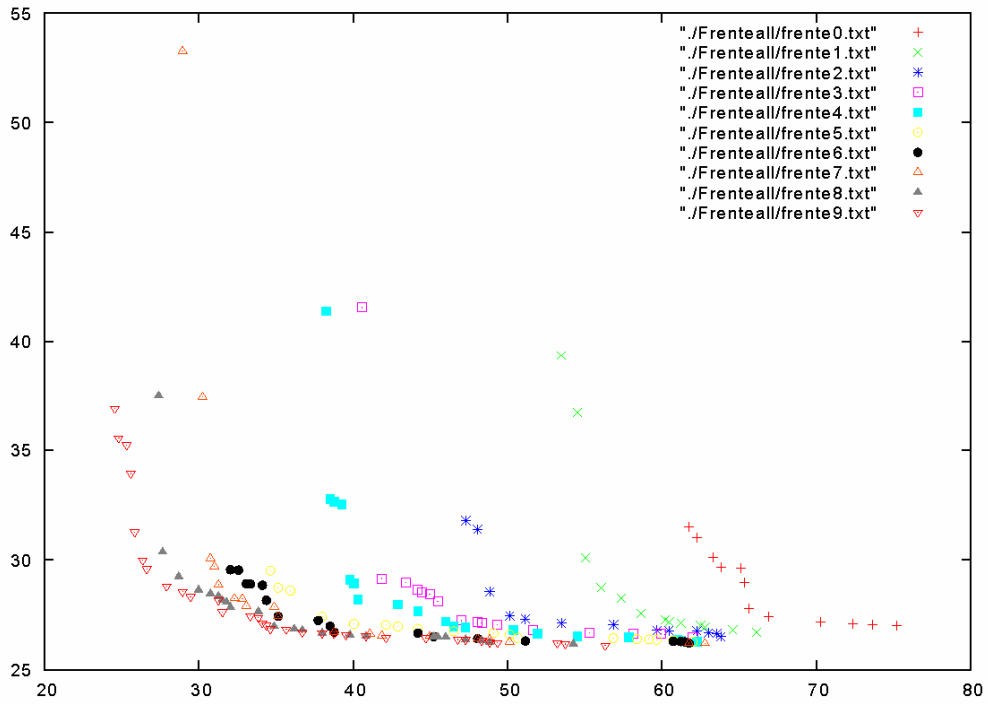


Ilustración 35. BCI-387-2PS: Evolución frente Prueba 5

También se muestran en la figura 36 los frentes finales para los conjuntos de entrenamiento, test y validación y en el que el mejor resultado tiene aproximadamente un 30% de los atributos y en validación con un resultado para test de un 26.9% y para validación de un 30.4% de fallos. Con lo que los resultados siguen estando en torno a los mismos valores que todas las pruebas anteriores.

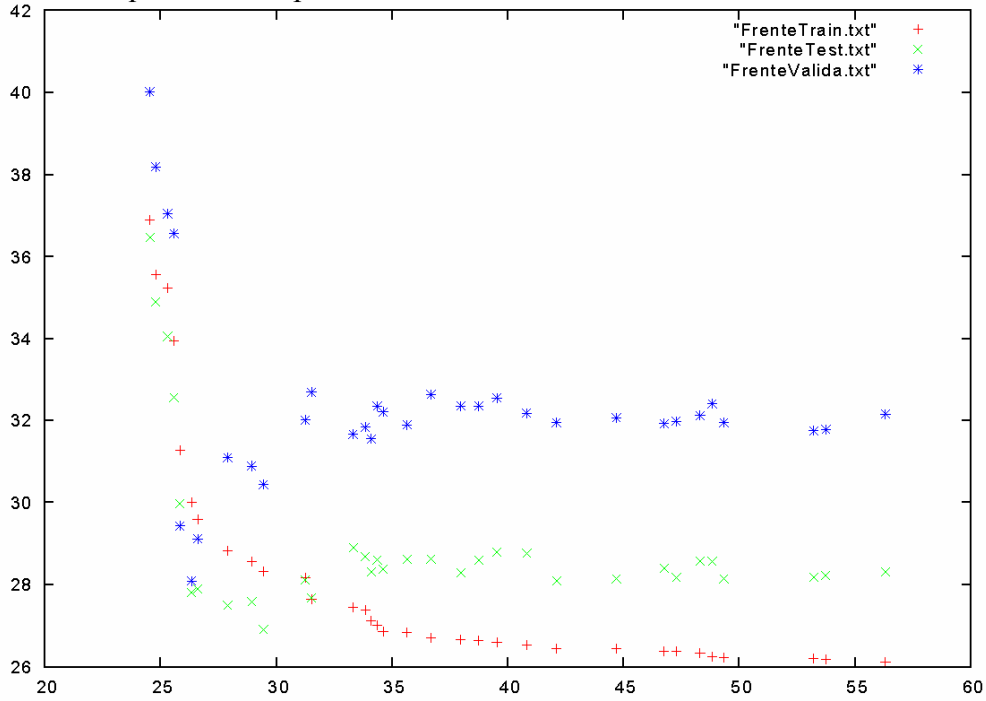


Ilustración 36. BCI-387-2PS: Frente final prueba 5

6.3.1.6.Prueba 6

Para esta prueba seguiremos inicializando los individuos al 80% de 1's y además bajaremos las generaciones que vimos en la prueba 4 que podría ofrecer un mejor resultado. Por ello, la población serán 48 individuos y 50 generaciones, con 300 iteraciones para el perceptron. En la figura 37 se puede ver la evolución de los frentes como simplemente se desplazan hacia la izquierda sin variar apenas el resultado del porcentaje de fallos.

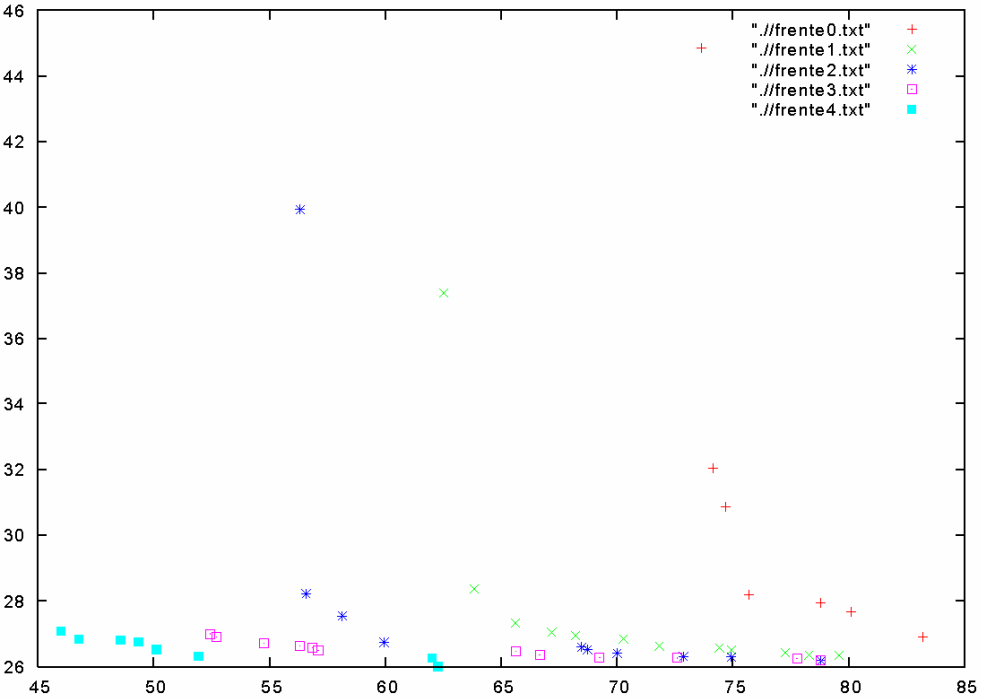


Ilustración 37. BCI-387-2PS: Evolución frente prueba 6

En la figura 38 se muestran los frentes finales que no son muy diferentes a cualquiera de las pruebas anteriores, de hecho el mejor resultado es de un 26.5% para test y un 30.3% en validación, lo que resulta muy parecido en comparación con las pruebas anteriores.

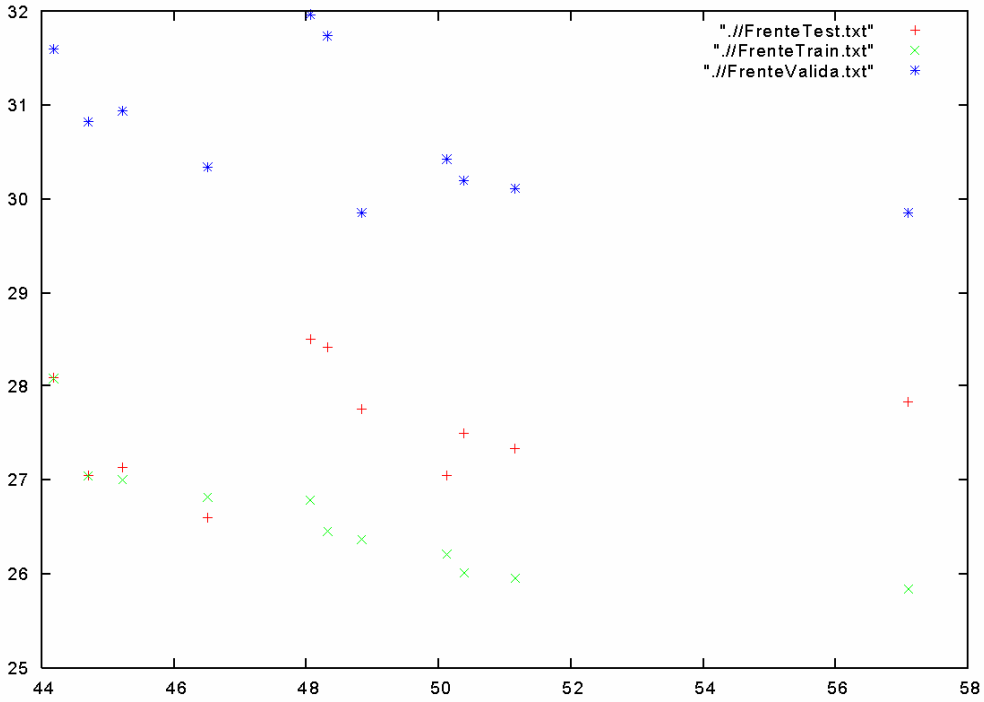


Ilustración 38. BCI-387-2PS: Frente final prueba 6

6.3.1.7.Prueba 7

Para esta prueba se decidió realizar una prueba estándar para ver si los resultados se parecían también a todas las pruebas anteriores. Para ello, la población será de 100 individuos, las generaciones de 100 y 300 iteraciones en el perceptron. En la figura 39 se muestra la evolución de los frentes.

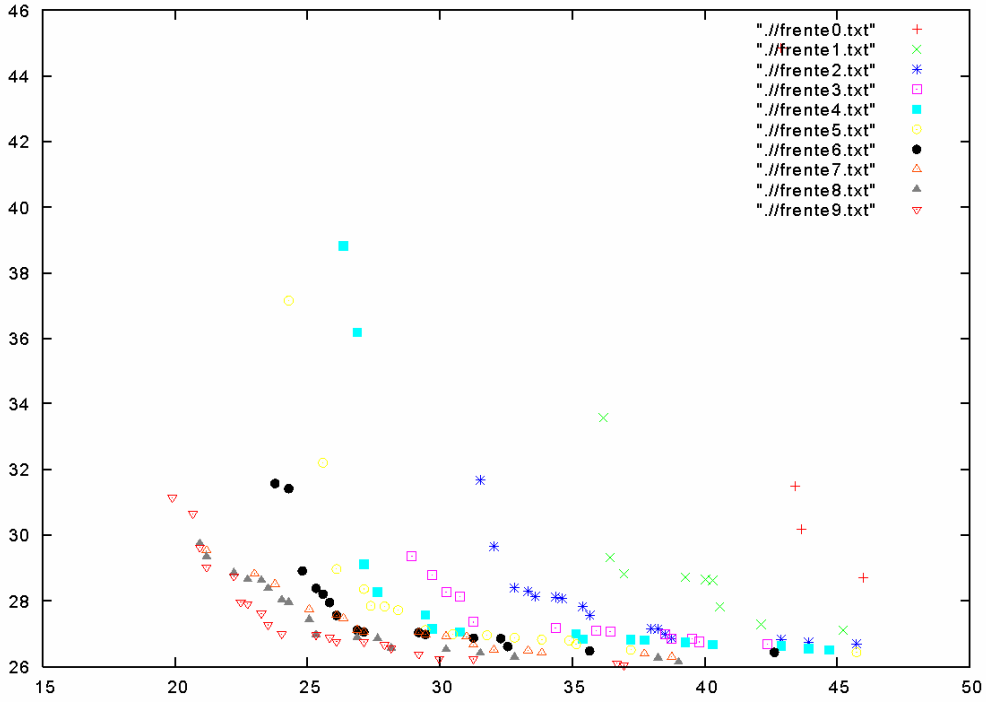


Ilustración 39. BCI-387-2PS: Evolución frentes prueba 7

En la figura 40 se puede observar como estos frentes finales son parecidos a los de las pruebas anteriores y los resultados son de hecho también similares. Para test el porcentaje de fallos es de un 27.3% y para validación de un 30.7%.

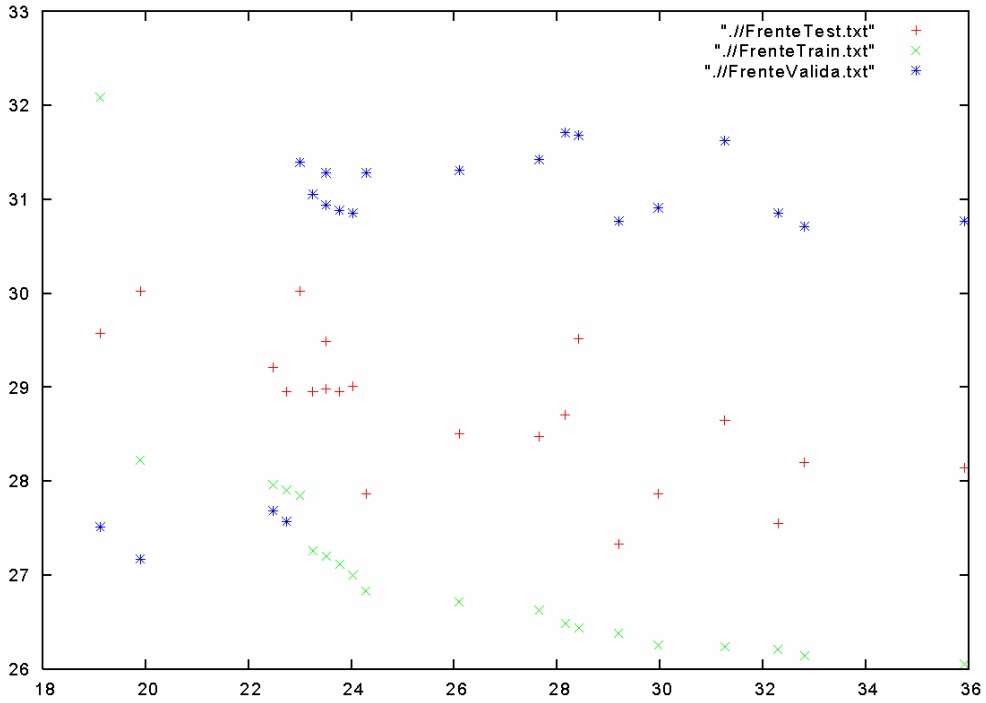


Ilustración 40. BCI-387-2PS: Frente final prueba 7

6.3.1.8.Prueba 8

Para esta prueba se decidió comprobar si los resultados seguirían siendo parecidos, aunque el número de iteraciones del perceptron fuera menor, y con ello consiguiéramos bajar el tiempo de ejecución de la prueba. Para ello, se eligieron 48 individuos de población y 50 generaciones, con 100 iteraciones para el perceptron. Además la inicialización de los individuo se realiza con un 80% de probabilidad de 1's como se puede ver en la evolución del frente de la figura 41.

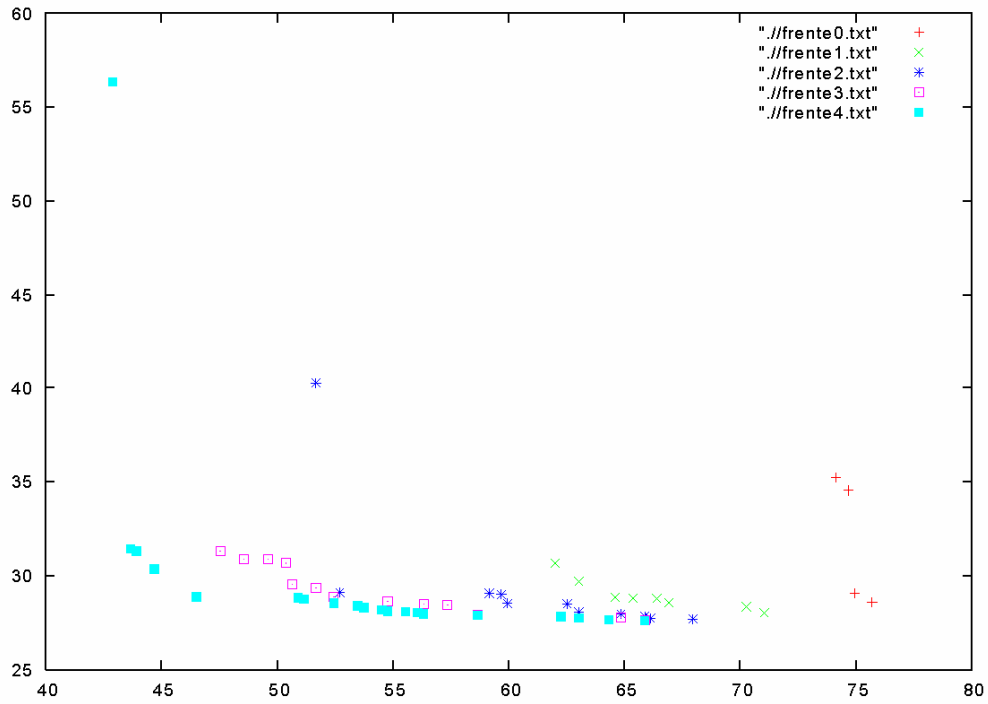


Ilustración 41. BCI-387-2PS: Evolución frente prueba 8

Como se puede ver claramente en la figura 42 el punto con un 50% de atributos tiene el mejor porcentaje de test con aproximadamente un 26.3% y en validación tiene un 31.7% de fallos. Con estos resultados, a pesar de que son ligeramente peores, se comprueba que merece la pena, debido a que el tiempo de ejecución fue bastante menor.

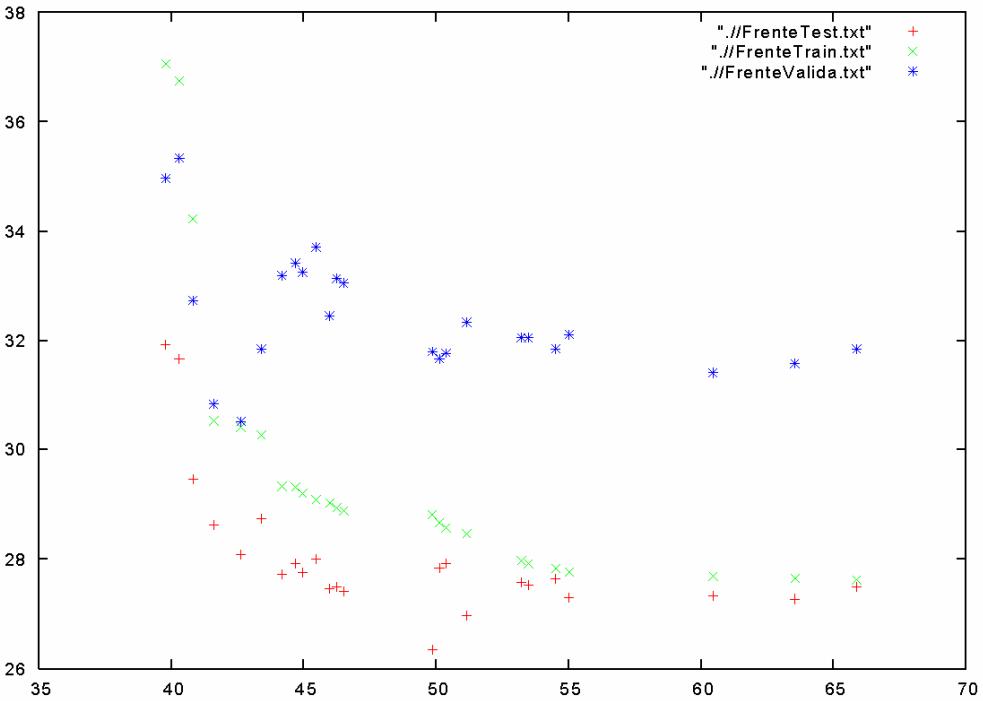


Ilustración 42. BCI-387-2PS: Frente final prueba 8

6.3.1.9.Prueba 9

En este caso se volvió a repetir la prueba 6, pero aquí el cambió consistió en inicializar los individuos al 50% de probabilidad de 1's. Para ello, los individuos serán 48, con 50 generaciones y 300 iteraciones para el perceptron. En la figura 43 se observa la evolución del frente.

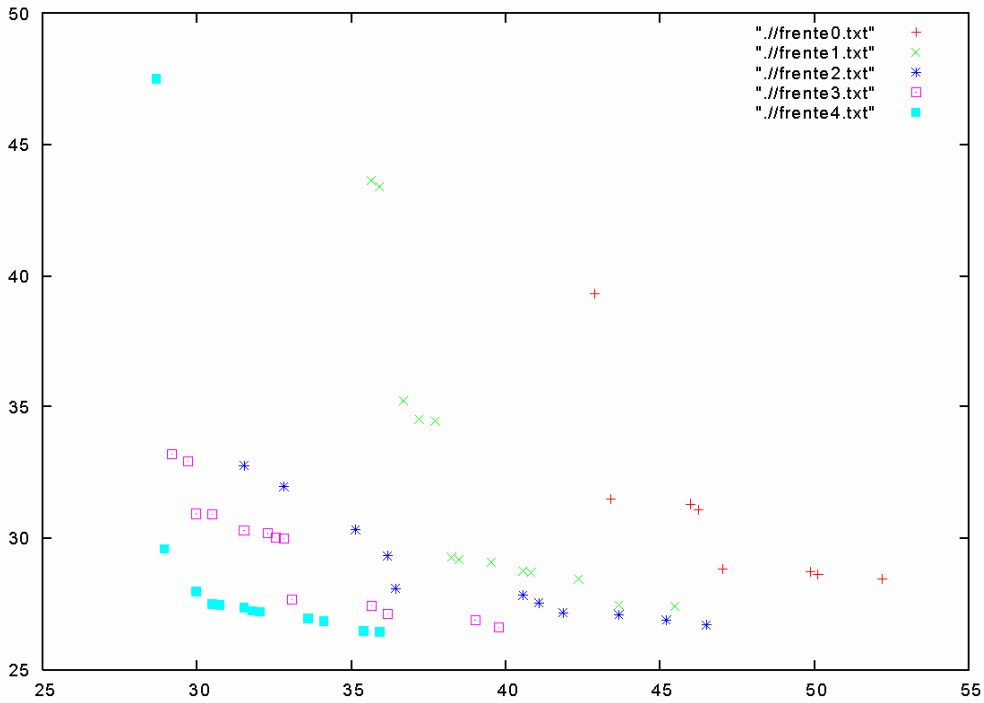


Ilustración 43. BCI-387-2PS: Evolución frente prueba 9

En este caso los resultados son bastante similares tanto a la prueba 6 como a la 8, que son las que se trataba de comparar, la única diferencia consiste en la cantidad de atributos de la mejor solución, ya que en este caso la solución que se obtiene tiene bastantes menos atributos como se puede ver en el punto con aproximadamente un 34.5% de atributos de la figura 44, y que corresponden con un error en test de un 26.3% y un 29.3% en validación.

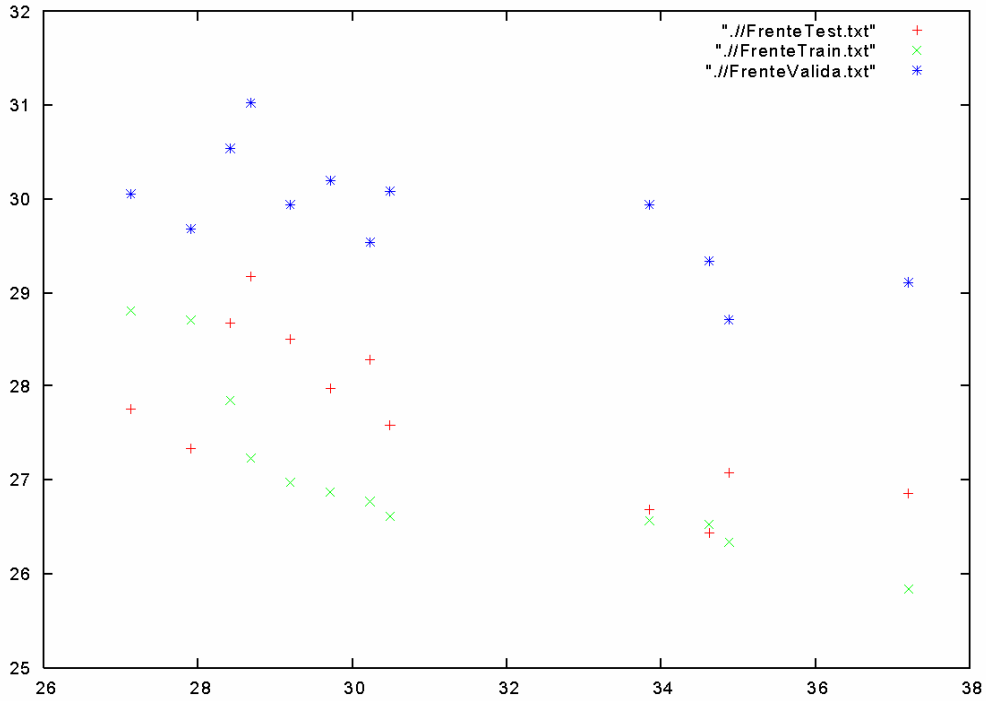


Ilustración 44. BCI-387-2PS: Frente final prueba 9

6.3.1.10. Prueba 10

Para esta prueba se decidió utilizar el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Se utilizó una población de 100 individuos con 100 generaciones y 300 iteraciones para el perceptron, la inicialización de los individuos será con una probabilidad del 50% de 1's. Como se puede ver en la figura 45 las escalas de los objetivos son ahora entre 0 y 100, por lo que se puede suponer que las tres clases se aprenden por igual. De hecho se observa como existen puntos muy cercanos al 100% de fallos para una clases y cerca del 0% en otra, pero en todos los casos, y como también existen puntos centrales en los que el aprendizaje esta compensado.

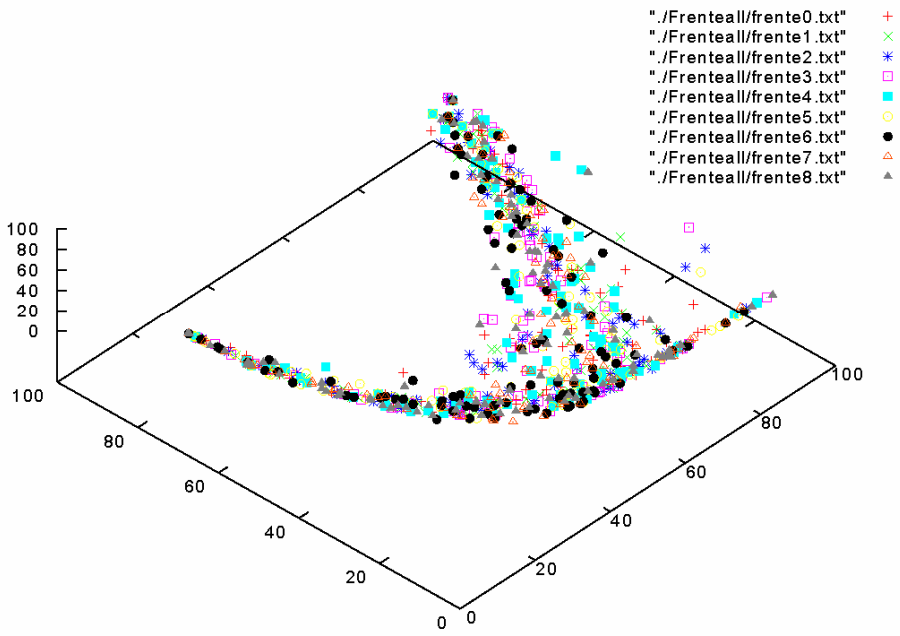


Ilustración 45. BCI-387-2PS: Evolución frente prueba 10

Para poder observar mejor la situación comentada anteriormente se realizó una comparación de los frentes dos a dos y que se puede ver en la figura 46, se puede ver como en este caso existen algunos frentes que describen una parábola comenzando en el punto (100,0) de la gráfica, pasando aproximadamente por el (50,50) y volviendo al (100, 100) y esto es debido a que algunos puntos aprenderán muy bien una clase en los dos casos extremos y también aprenderán las tres clases de forma semejante en el caso central. Sin embargo, también se puede observar como otros frentes no realizan esa parábola, para verlo en más detalle nos centraremos sólo en los frentes del conjunto de entrenamiento.

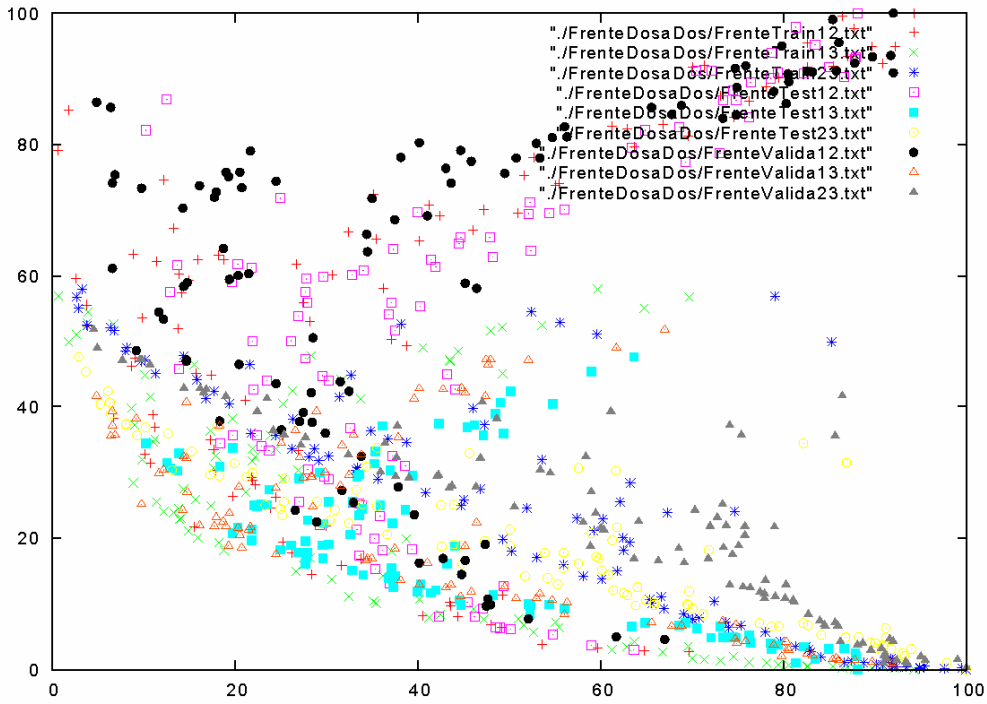


Ilustración 46. BCI-387-2PS: Comparación frentes dos a dos prueba 10

En la figura 47 se muestran la comparación de objetivos sólo para el conjunto de entrenamiento, y ahora se puede ver con más detalle como para el caso de la comparación 1 y 2 si se produce el hecho de la parábola, mientras que para los otras dos comparaciones no. En cualquier caso, para los tres frentes se observa que al intentar disminuir el error de una clase se incrementa el error de la otra, siendo este efecto más acusado en el caso de las clases dos y tres.

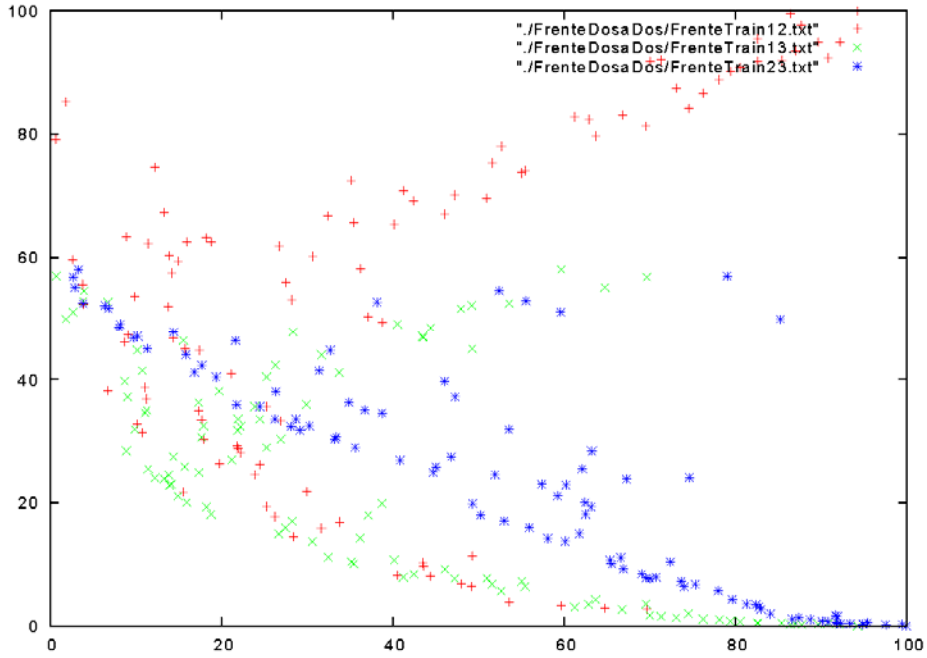


Ilustración 47. BCI-387-2PS: Comparación frenteTrain dos a dos prueba 10

En la figura 48 se muestra el frente final para los conjuntos de entrenamiento, test y validación y no permite observar cual es el mejor punto del frente, aunque se trata del punto que tiene para cada una de las clases 311, 866 y 975 fallos respectivamente en entrenamiento, y obtiene un porcentaje de fallos de 26.51% en test y 31% en validación con lo que se puede decir que utilizar este caso con estos objetivos no varía mucho el resultado en comparación con los resultados obtenidos con el primer caso

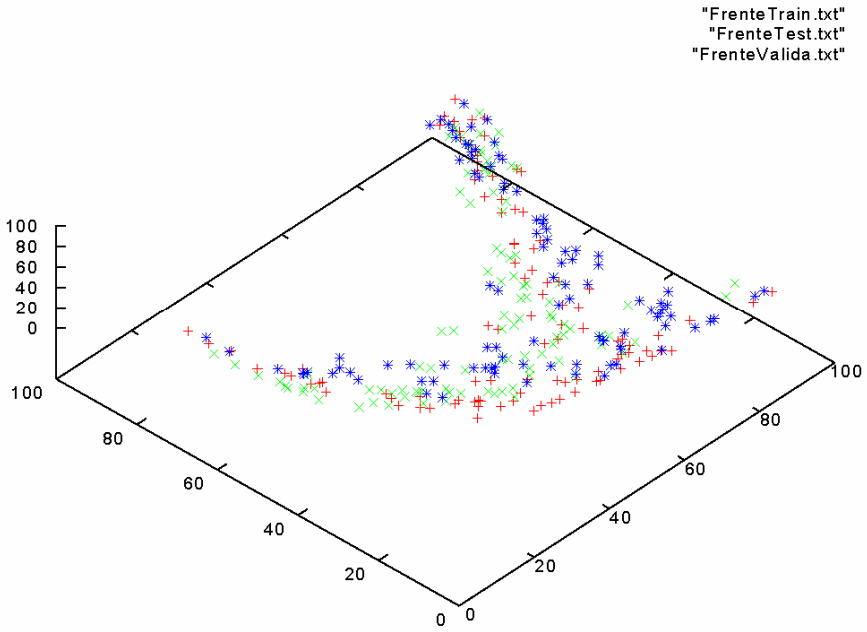


Ilustración 48. BCI-387-2PS: Frente final prueba 10

6.3.1.11. Prueba 11

Para la presente prueba se decidió continuar con el segundo caso, pero se decidió aumentar la población ya que en las pruebas realizadas sobre los 387 atributos sin desordenar el fichero de entrenamiento, daba un mejor resultado. Para ello se utilizaron 200 de población y 50 generaciones, dejando los demás parámetros igual que en la prueba anterior. En la figura 49 se presenta la evolución del frente, aunque lo único que se puede distinguir es la escala que se encuentra entre 0 y 100 para las tres clases.

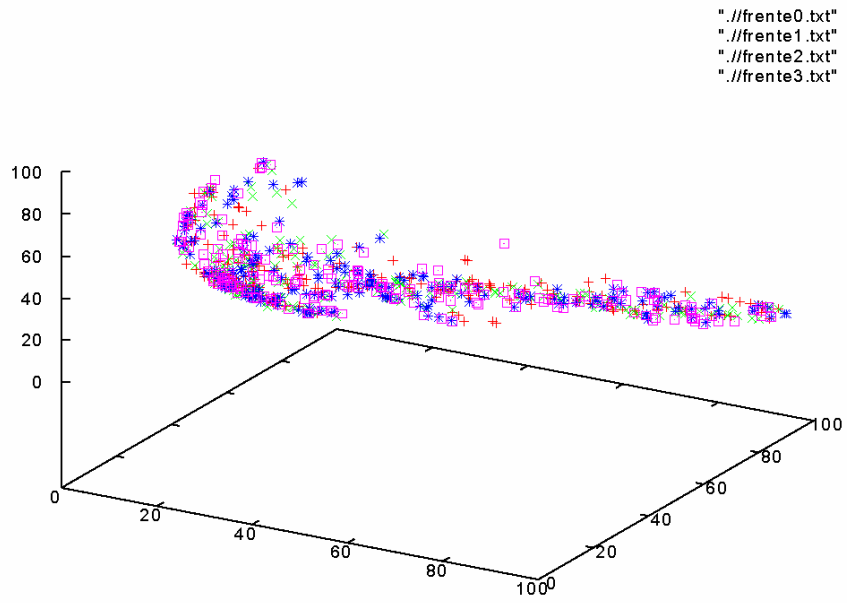


Ilustración 49. BCI-387-2PS: Evolución frente prueba 11

En la figura 50 se muestra la comparación de los frentes dos a dos, y en ella se parece intuir prácticamente los mismos frentes finales, pero con muchos más puntos que la prueba anterior.

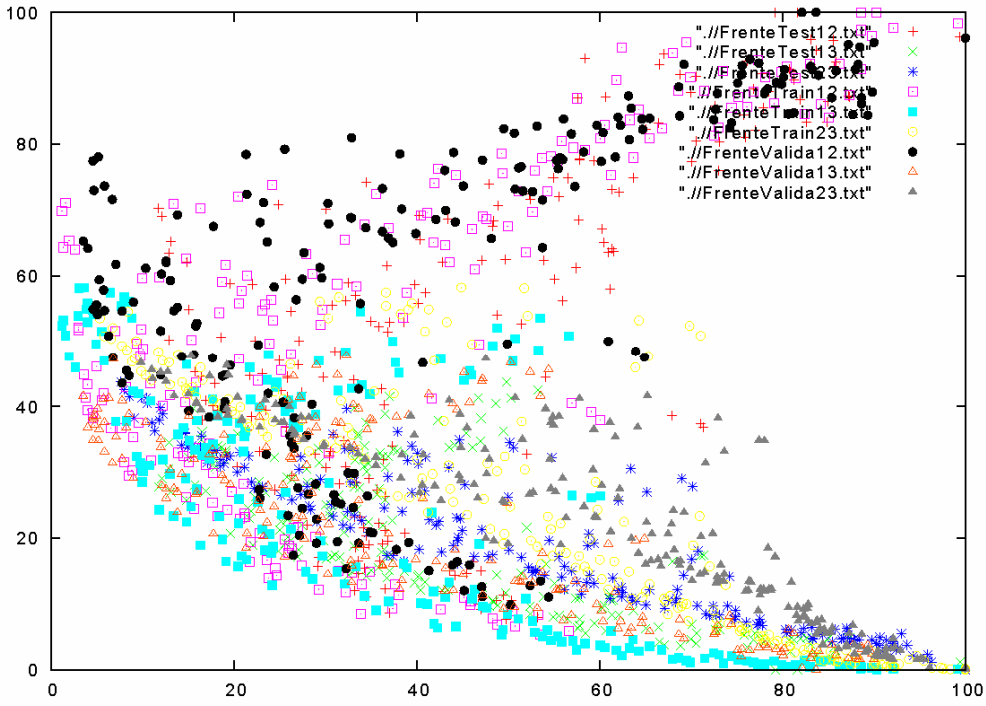


Ilustración 50. BCI-387-2PS: Comparación frentes dos a dos prueba 11

En la figura 51 se ve que es prácticamente similar a la figura 47, pero que esta cuenta con muchísimos más puntos en el frente dominante, esto quiere decir que el utilizar una población mayor, si bien no cambiará el resultado final en gran medida, si ha conseguido que el frente final obtenga muchos más puntos.

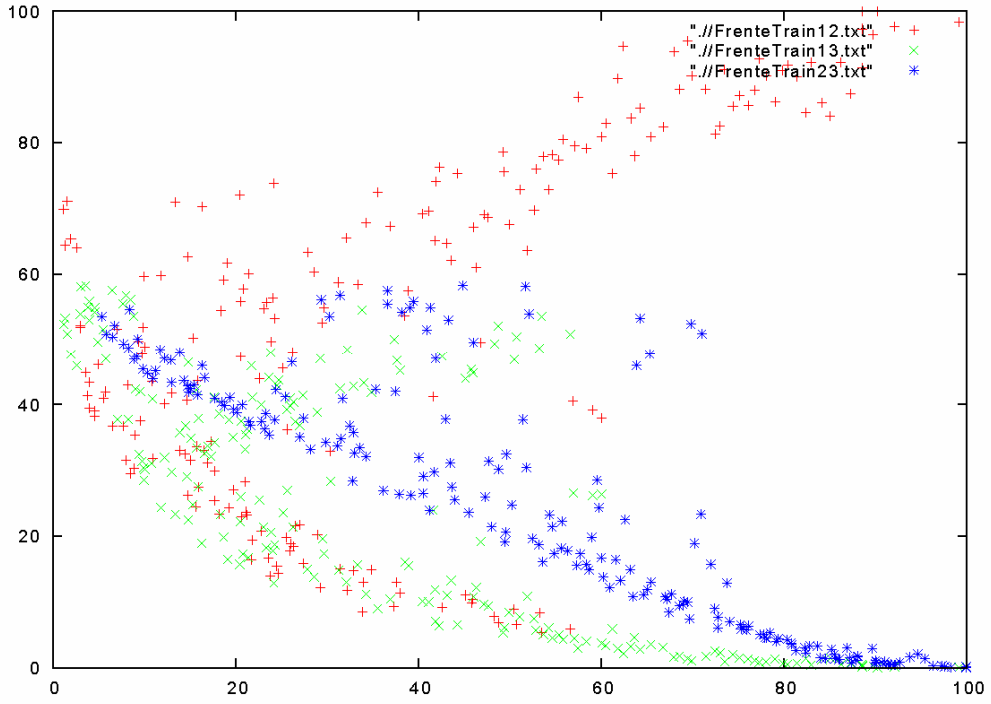


Ilustración 51. BCI-387-2PS: Comparación frente entrenamiento dos a dos prueba 11

En la figura 52 se muestra el frente final para los tres conjuntos de datos, aunque no es fácil ver cuál sería el mejor punto para test, que tiene 327, 578 y 1037 fallos para cada una de las clases, con un 26.54 en test y un 30,4 en validación. Se puede ver en este caso, como también es la clase 3 la que obtiene un peor resultado y como efectivamente los resultados son prácticamente iguales que en la prueba anterior, aunque tiene muchos más puntos el frente.

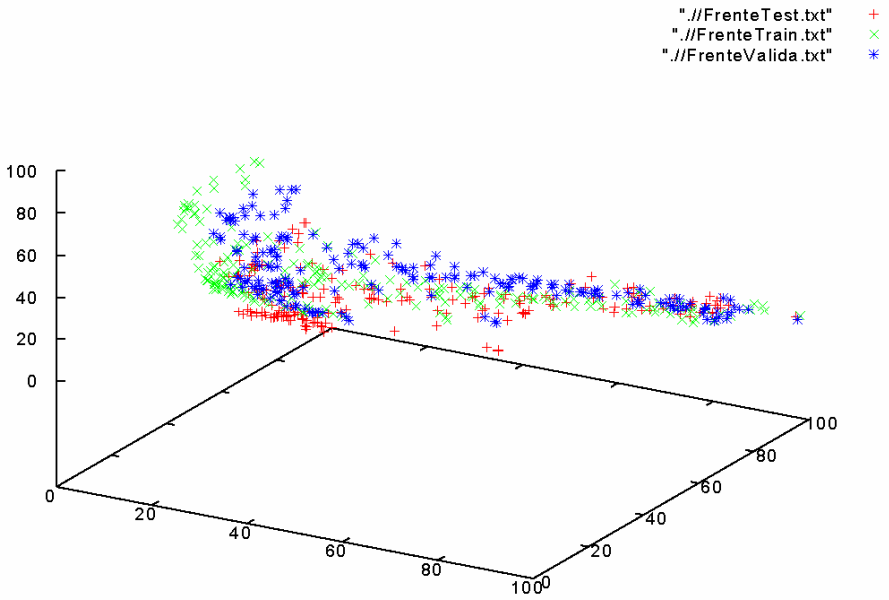


Ilustración 52. BCI-387-2PS: Frente final prueba 11

6.3.1.12. Prueba 12

Se realizó otra prueba más en con el segundo caso, pero en este caso se realizó una inicialización de los individuos con el 80% de 1's, siendo en este caso la población 148 individuos, con 100 generaciones y 100 iteraciones para el perceptron. En la figura 53 se muestra la evolución de los frentes en la que se puede ver como la escala en este caso para la clase dos no sube del 60% y para la clase 1 del 90%, esto se debe a que no se han encontrado soluciones en el frente en las que los valores suban de esos porcentajes.

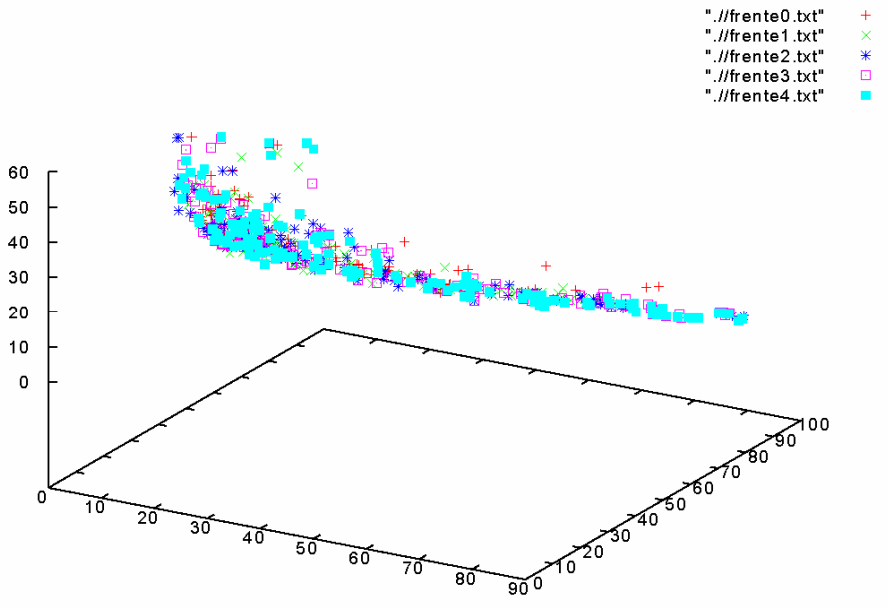


Ilustración 53. BCI-387-2PS: Evolución frente prueba 12

En la figura 54 se muestra la comparación de los frentes dos a dos, de manera que no se aprecian muchos cambios con la prueba anterior, y por tanto, las conclusiones deberían ser las mismas, que los resultados serán similares y que la clase obtendrá el peor resultado.

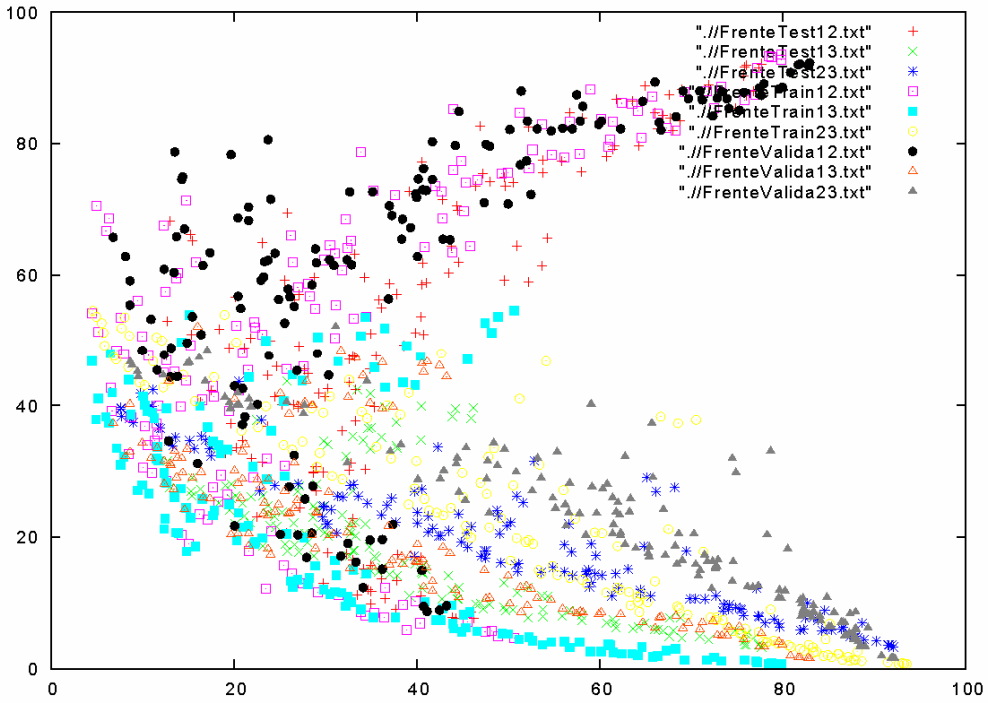


Ilustración 54. BCI-387-2PS: Comparación frentes dos a dos prueba 12

En la figura 55 se muestra el frente final en la que al no apreciarse el mejor punto, este tiene 190, 843 y 1050 fallos para cada una de las clases con un 26.34% para test y un 30.5% para validación, confirmando los supuestos realizados anteriormente.

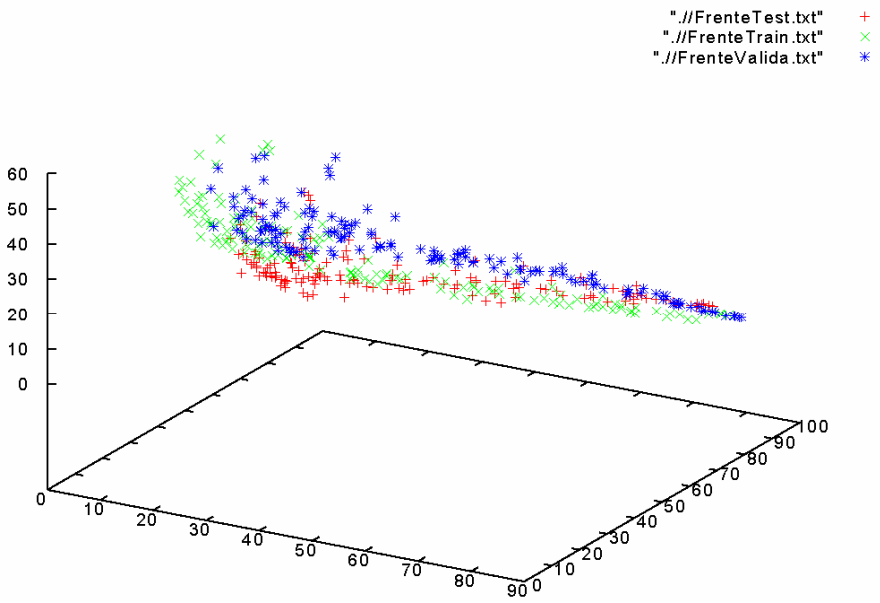


Ilustración 55. BCI-387-2PS: Frente final prueba 12

6.3.1.13. Resultados finales

En la tabla 9 se presentan los resultados de todas las pruebas realizadas con 387 atributos seleccionados y con el fichero de entrenamiento con los datos desordenados, y en los que se variaron los parámetros que aparecen en negrita y que serían:

- Caso: Se refiere a los objetivos que se utilizaron; caso 1, minimizar el número de atributos y el número de fallos total; caso 2, minimizar el número de fallos para cada una de las clases.
- Pob.: Se refiere al número de individuos que contendrá la población del algoritmo genético multiobjetivo.
- Gen.: Se refiere al número de generaciones que se ejecutará el algoritmo genético multiobjetivo.
- Ite.: Se refiere al número de ciclos de aprendizaje que realizaran los perceptrones simples.
- Prob. 1's: Se refiere a la probabilidad de 1's con la que se inicializarán los individuos y su valor se encuentra entre 0 y 1.
- Inicial: Indica si se ha inicializado con individuos por fichero de otras pruebas anteriores. No se ha inicializado ninguna de las pruebas realizadas así que no ha incluido este valor en la tabla 9.

Prueba	**Caso**	**Pob.**	**Gen.**	**Ite.**	**Prob. 1's**	**Obj0**	**Obj1**	**Obj2**	**%Tot.**	**%Test**	***%Val***
1	**1**	**32**	**4**	**300**	**0.5**	184	2017	-	28.98	28.1	*28.5*
2	**1**	**100**	**100**	**300**	**0.5**	43	1999	-	28.7	26.4	*32.5*
3	**1**	**48**	**200**	**300**	**0.5**	70	1924	-	27.6	26.9	*32.1*
4	**1**	**100**	**30**	**300**	**0.5**	109	1937	-	27.8	26.7	*29.5*
5	**1**	**100**	**100**	**300**	**0.8**	114	1971	-	28.3	26.9	*30.4*
6	**1**	**48**	**50**	**300**	**0.8**	180	1866	-	26.8	26.5	*30.3*
7	**1**	**100**	**100**	**300**	**0.5**	113	1836	-	26.3	27.3	*30.7*
8	**1**	**48**	**50**	**100**	**0.8**	193	2005	-	28.8	26.3	*31.7*
9	**1**	**48**	**50**	**300**	**0.5**	134	1846	-	26.5	26.3	*29.3*
10	**2**	**100**	**100**	**300**	**0.5**	311	866	975	30.92	26.51	*31*
11	**2**	**200**	**50**	**300**	**0.5**	327	578	1037	27.9	26.54	*30.4*
12	**2**	**148**	**50**	**100**	**0.2**	190	843	1050	29.92	26.34	30.5

Tabla 9. Resultados BCI con 2PS

Se puede ver como todas las pruebas a pesar de realizar combinaciones de casos, población, generaciones e inicialización del perceptron son prácticamente iguales, de hecho no existe más de un 3% de diferencia entre el mejor resultado y el peor. De hecho el mejor resultado obtenido es sorprendentemente el de la primera prueba, que se realizó con una población y unas generaciones muy bajas, obteniendo un 71.5% de aciertos para el conjunto validación, por lo que las diferencias observadas en la tabla 9 no son significativas estadísticamente.

Y ahora si nos centramos en las pruebas, vemos como el utilizar el segundo caso ofrece siempre más o menos los mismos resultados, a pesar incluso de que se bajen los ciclos de aprendizaje de los perceptrones como se ve en la última prueba.

Sin embargo para el primer caso, ocurre un hecho bastante curioso, y es que al disminuir las generaciones sobre al algoritmo genético multiobjetivo los resultados sobre la validación son mejores, muy ligeramente pero mejores. Ya que aunque no sea muy importante, existe una diferencia de un 4% aproximadamente entre el mejor y el peor resultado en validación.

También cabe destacar si comparamos las pruebas 6, 8 y 9 podemos ver como el utilizar unas iteraciones de los perceptrones menor si comparamos la prueba 8 con la 6 el resultado que se obtiene es algo peor, aunque el tiempo de ejecución se reduce considerablemente. Si comparamos ahora la prueba 6 con la 9, vemos como ampliar el espacio de búsqueda inicializando los individuos con más probabilidad de 1's empeora el resultado ligeramente también. Sin embargo si ahora comparamos la prueba 2 con la 5 vemos como este resultado es mejor al ampliar el espacio de búsqueda. Esto puede ser debido a que ampliar el espacio de búsqueda puede requerir de más generaciones para llegar a un resultado correcto, y que con pocas generaciones los resultados que se consiguen son peores.

Si ahora comparamos la prueba 2 con la 7, observamos como tienen los mismos parámetros y sin embargo los resultados son distintos. Con estas pruebas se pretendía comprobar cómo la ejecución de este software es estocástica y dos ejecuciones como los mismos parámetros de entrada pueden producir dos salidas, y por tanto, dos resultados distintos.

En cualquier caso y como ya se ha comentado, la primera conclusión es que las distintas configuraciones utilizadas no ofrecen variaciones apreciables y están alrededor del 30% de error. En segundo lugar, hay que recordar que la combinación de dos perceptrones simples con 387 atributos conseguía un 33.65% de error (66.35% de aciertos). En este sentido, el sistema realizado si ofrece cierta mejora frente al clasificador base en todos los casos siendo aproximadamente un 5% de mejora en el mejor caso.

Con el resultado de un 71.5% de aciertos en el conjunto de validación si los comparamos con los de la competición sobre el sujeto uno, http://ida.first.fraunhofer.de/projects/bci/competition_iii/results/index.html si bien nuestra posición en la competición estaría rondando el puesto 15 sobre el sujeto 1, nos aproximamos bastantes a los resultados, ya que por ejemplo el segundo mejor para el sujeto 1 tiene un 79% de aciertos y el 10º un 72%, con lo que se puede decir que si bien

los resultados no son espectaculares, si que estamos en los resultados típicos obtenidos en esa competición superando el 70% en algunos casos.

6.3.2. Pruebas con 387 atributos con tres perceptrones simples

Viendo los resultados obtenidos sobre la combinación de tres perceptrones simples explicadas en el apartado 2.2.2 "Utilización de tres perceptrones simples", que eran mejores que el uso de dos perceptrones (a costa de un mayor tiempo), se decidió realizar la integración de estos tres perceptrones en el software realizado, ya que si recordamos los resultados sobre dos perceptrones simples se obtenía un 66.35% de aciertos mientras que con tres perceptrones simples se obtiene un 71.51%. Así mismo y teniendo en cuenta que los resultados obtenidos eran prácticamente similares para los casos realizados, se decidió realizar esta implementación solamente para el primero de los casos. Para realizar estas pruebas será necesario tener en cuenta los ciclos de aprendizaje de los perceptrones, ya que como observamos en las pruebas más simples, este tiempo era superior que con la combinación de dos perceptrones. Por ello, se decidió utilizar también lo explicado en el apartado 5.6 "Algunas consideraciones adicionales" de manera que las iteraciones fueran aumentando linealmente conforme avanzará la ejecución del algoritmo. Además en este caso, se decidió también que para devolver el valor sobre el frente dominante de la última generación del algoritmo el número de ciclos de aprendizaje pudiera ser diferente al utilizado en la ejecución normal del algoritmo y por ello, se podrá cambiar este valor. Es decir, para afinar mejor el porcentaje de aciertos obtenido por el frente final, se utilizarán muchos ciclos de aprendizaje sin que esto tenga ninguna influencia sobre la fase de aprendizaje.

6.3.2.1.Prueba 1

Lo primero que se quiso comprobar es que efectivamente esta integración podía obtener resultados como mínimo similares a las pruebas simples, y para ello se escogieron individuos de pruebas anteriores y se decidió observar cual era su comportamiento. La población será de 24 individuos con sólo una generación y las iteraciones máximas y con las que se conseguirá el frente final será de 500, siendo 0.5 la razón de aprendizaje.

Como esta prueba sólo tenía una generación no es viable ver la evolución de los frentes y por tanto sólo se representa el resultado final para los conjuntos de entrenamiento, test y validación en la figura 56. En ella se puede ver como los puntos escogidos eran de diferentes pruebas anteriores y por tanto, el número de atributos era muy dispar, el mejor resultado para test se encuentra en el punto más a la derecha de la gráfica y tiene un 28.5% de fallos aproximadamente para validación.

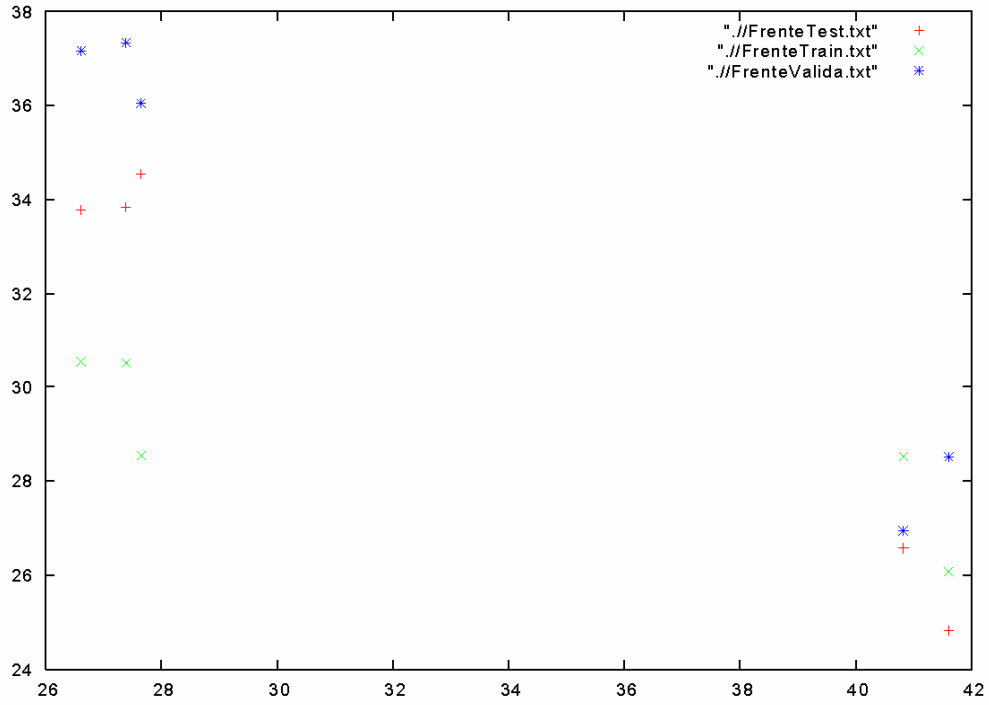


Ilustración 56. BCI-387-3PS: Frente final prueba 1

6.3.2.2.Prueba 2

Se realizó un prueba bastante sencilla en la que se los individuos fueron 48 y con 10 generaciones y las iteraciones máximas y con las que se conseguirá el frente final será de 500, siendo 0.8 la razón de aprendizaje.

Como esta prueba sólo tenía diez generaciones ver la evolución de los frentes tampoco aportaría muchas conclusiones, con lo que sólo se representa el resultado final para los conjuntos de entrenamiento, test y validación en la figura 57. Se puede observar como los puntos del frente dominante van bajando de atributos, y como los resultados son peores que la prueba anterior. Esto puede ser debido al cambio realizado en la razón de aprendizaje. El mejor punto del frente dominante también es el que se encuentra más a la derecha y tiene un 29% de fallos en validación aproximadamente.

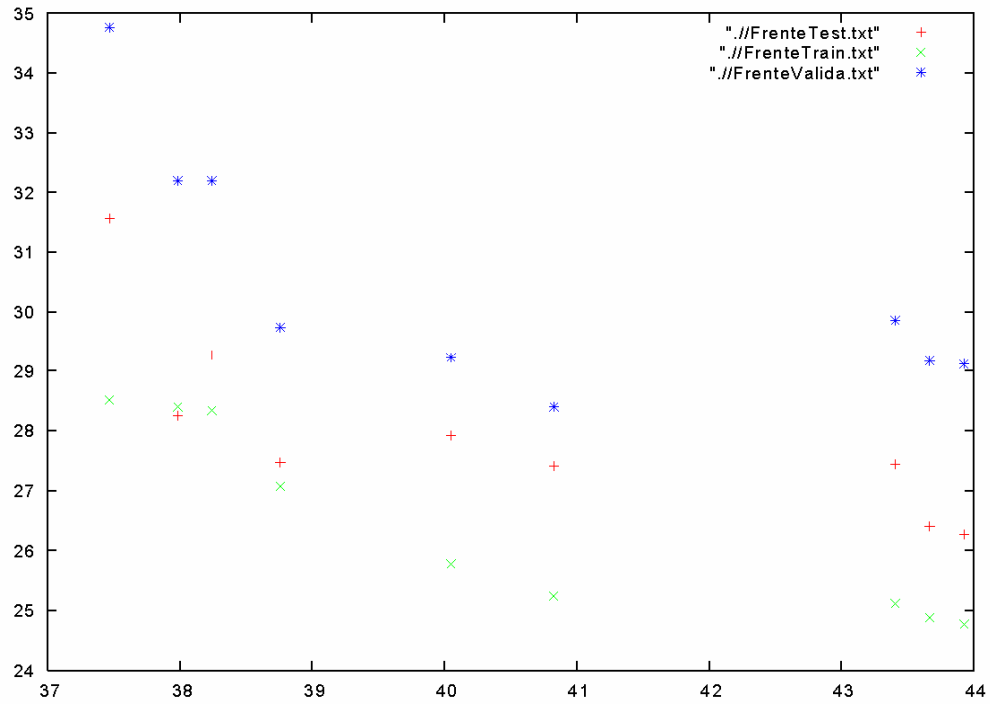


Ilustración 57. BCI-387-3PS: Frente final prueba 2

6.3.2.3.Prueba 3

Se realizó otra prueba sencilla en la que se los individuos fueron 48 y con 10 generaciones y las iteraciones máximas serán 200 y con las que se conseguirá el frente final serán 1000, siendo 0.5 la razón de aprendizaje. El principal cambio se encuentra en la razón de aprendizaje.

Por la misma razón que la prueba anterior, sólo se representa el resultado final para los conjuntos de entrenamiento, test y validación en la figura 58. Se puede observar como los puntos del frente dominante van bajando de atributos y como por tanto se puede deducir que se necesitan más generaciones para encontrar un mejor frente, es decir, que contenga puntos también con un porcentaje bajo de atributos. Comparando estos resultados con los de la prueba anterior, se mantendrá la razón de aprendizaje a 0.5 para las siguientes pruebas. El mejor punto del frente dominante tiene aproximadamente un 39% de atributos y por debajo del 27% de fallos en validación.

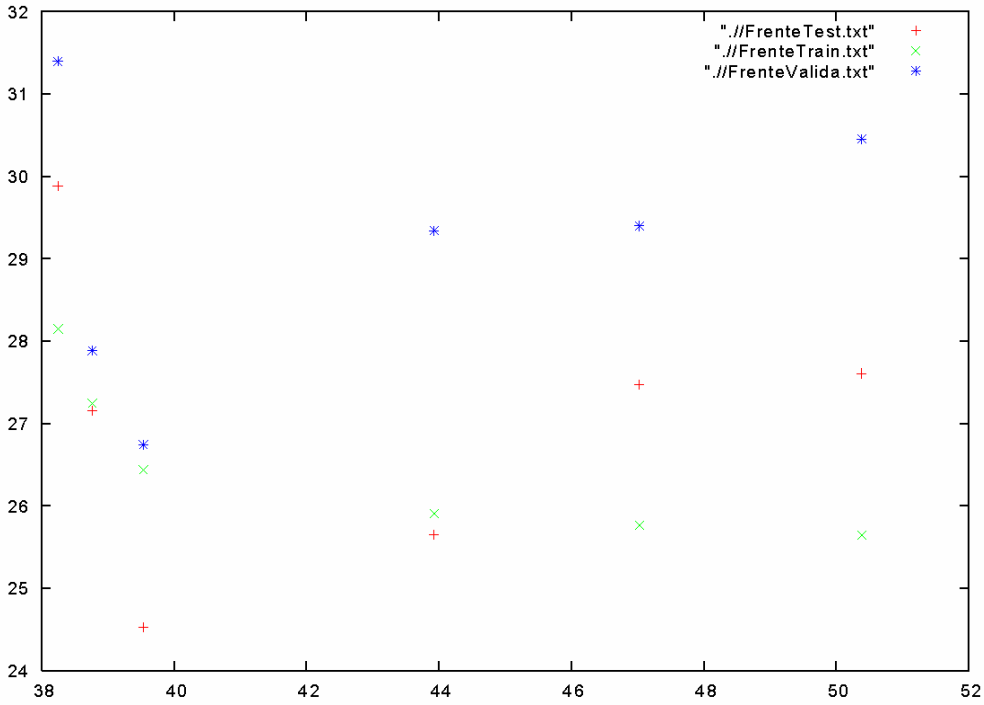


Ilustración 58. BCI-387-3PS: Frente final prueba 3

6.3.2.4.Prueba 4

Para esta prueba se decidió aumentar el número de generaciones y la población siendo 100 en los dos casos, para que el tiempo de la prueba no aumente demasiado, se decidió bajar las iteraciones máximas a 100 y con las que se conseguirá el frente final serán 1000, siendo 0.5 la razón de aprendizaje.

En la figura 59 se puede observar como los frentes mantienen prácticamente los mismos resultados en el eje Y, mientras que para el eje X que representa el porcentaje de atributos nos damos cuenta cómo va bajando. Esto significa que los resultados no mejoran en gran medida y lo que ocurre es que se encuentran soluciones con menos atributos capaces de obtener el mismo resultado que los individuos con un mayor número de atributos.

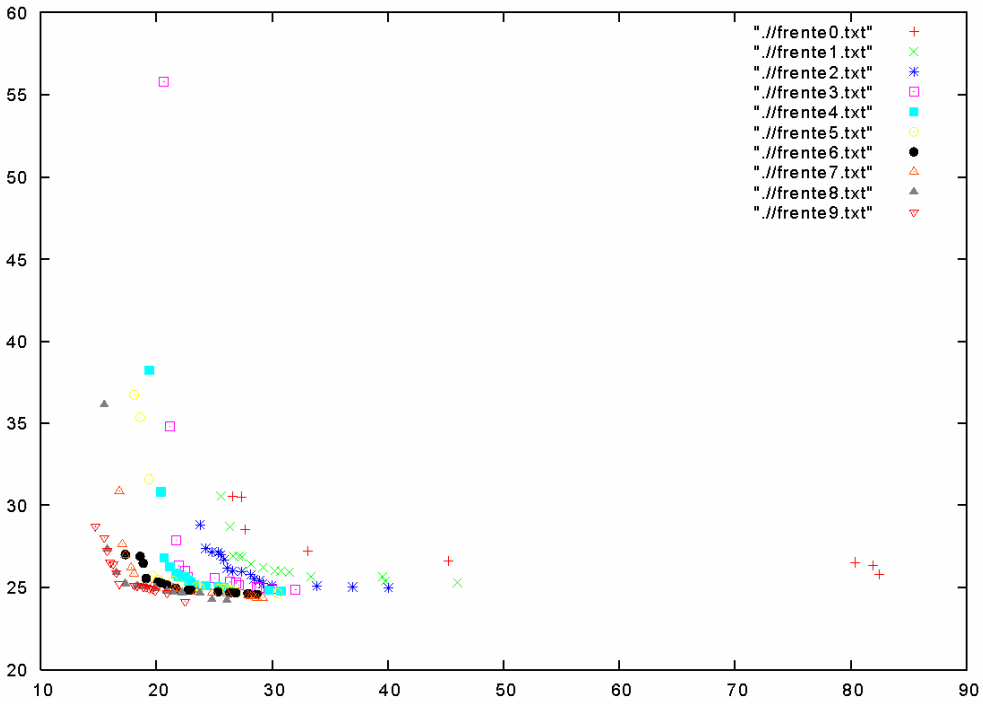


Ilustración 59. BCI-387-3PS: Evolución frente prueba 4

En la figura 60 se puede apreciar como el mejor punto de test se encuentra aproximadamente con un 16% de los atributos y el resultado para test es de aproximadamente un 25% de fallos, sin embargo en la mayoría de los casos existe un diferencia de un 5% aproximadamente entre el resultado de test y validación y como el resultado para validación es de un 29% de fallos aproximadamente.

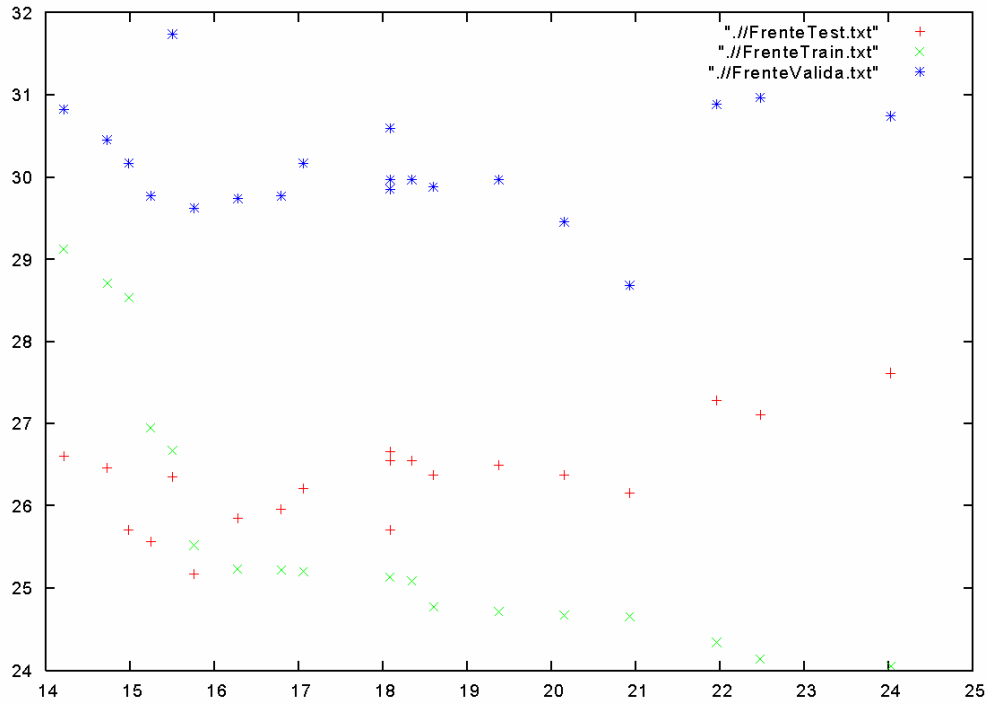


Ilustración 60. BCI-387-3PS: Frente final prueba 4

6.3.2.5.Prueba 5

Para esta prueba se decidió bajar el número de generaciones y también aumentar los ciclos de aprendizaje máximos, basándonos en la experiencia de las pruebas realizadas anteriormente y también para intentar que la diferencia entre entrenamiento y validación se redujera. La población será de 48 individuos y 50 generaciones, las iteraciones máximas serán 300 y con las que se conseguirá el frente final serán 500, siendo 0.5 la razón de aprendizaje.

En la figura 61 se puede observar la evolución del frente que al principio se observa como mejora en los dos ejes, pero luego se observa como los resultados simplemente mejoran en el porcentaje de atributos usados, al igual que en la prueba anterior.

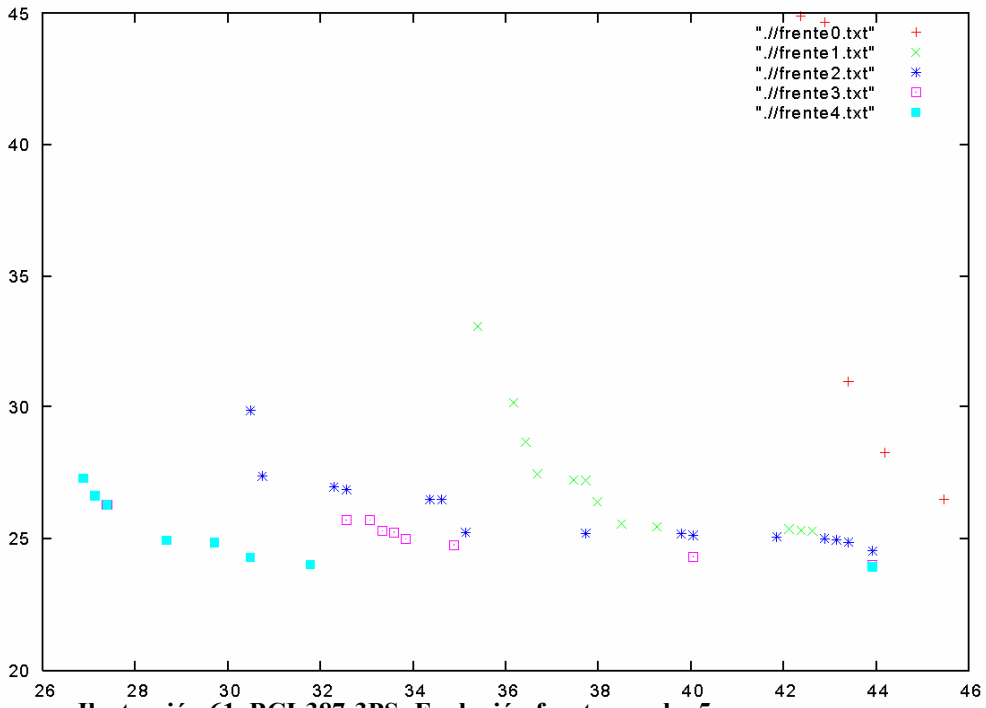


Ilustración 61. BCI-387-3PS: Evolución frente prueba 5

En la figura 62 se puede observar el frente final para los conjuntos de entrenamiento, test y validación y como existe también esa diferencia de aproximadamente el 5% entre los resultados de los conjuntos de test y validación. El mejor punto para test es el que tiene aproximadamente un 44% de atributos y obtiene un resultado para validación del 29% aproximadamente.

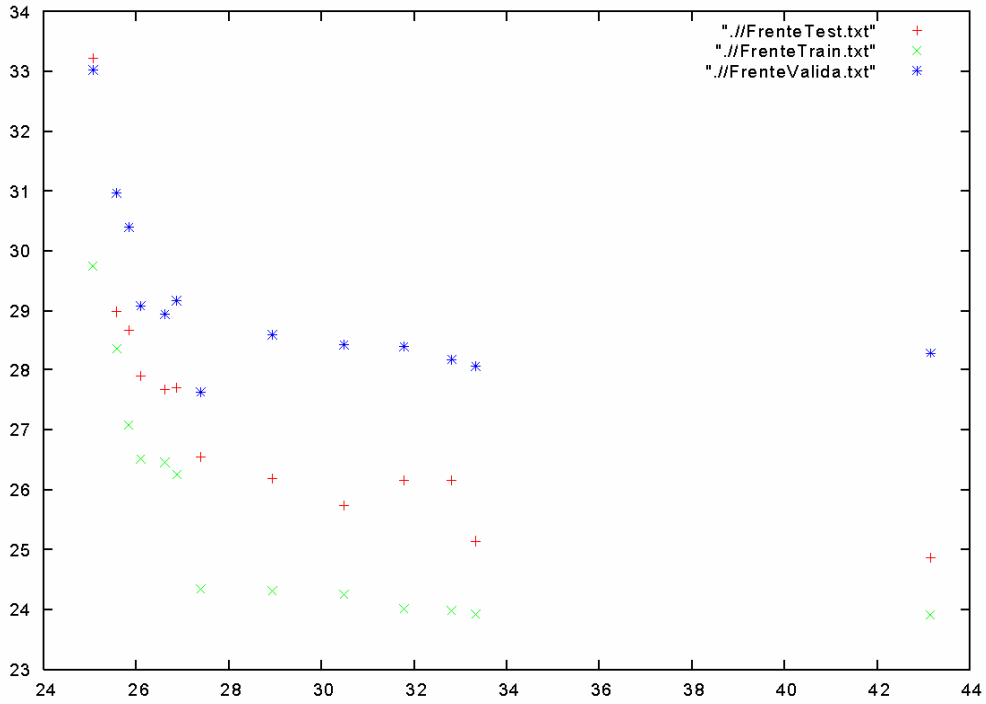


Ilustración 62. BCI-387-3PS: Frente final prueba 5

6.3.2.6.Resultados finales

En la tabla 10 se presentan los resultados de todas las pruebas realizadas con 387 atributos seleccionados y con el fichero de entrenamiento con los datos desordenados y utilizando como clasificador tres perceptrones simples combinados, todas realizadas para el primer caso, es decir, minimizar el número de atributos y el número de fallos total, y en los que se variaron los parámetros:

- Pob.: Se refiere al número de individuos que contendrá la población del algoritmo genético multiobjetivo.
- Gen.: Se refiere al número de generaciones que se ejecutará el algoritmo genético multiobjetivo.
- Ite.Máx: Se refiere al número de ciclos de aprendizaje máximo que realizaran los perceptrones simples.
- Ite.Val: Se refiere al número de ciclos de aprendizaje con el que se ejecutará el frente dominante de la última ejecución y que servirá para mostrar los resultados.
- α: Razón de aprendizaje de los perceptrones simples.
- Prob. 1's: Se refiere a la probabilidad de 1's con la que se inicializarán los individuos y su valor se encuentra entre 0 y 1.
- Inicial: Indica si se ha inicializado con individuos por fichero de otras pruebas anteriores.

Prueba	Pob.	Gen.	IteMax	IteVal	α	Prob. 1's	Inicial	Obj0	Obj1	%Tot	%Test	*%Val*
1	**24**	**1**	**500**	**500**	**0.5**	**-**	**SI**	161	1815	26	24.8	*28.5*
2	**48**	**10**	**500**	**500**	**0.8**	**0.5**	**NO**	170	1724	24.7	26.2	*29.1*
3	**48**	**10**	**200**	**1000**	**0.5**	**0.5**	**NO**	153	1840	26.4	24.5	*26.7*
4	**100**	**100**	**100**	**1000**	**0.5**	**0.5**	**NO**	61	1776	25.5	25.1	*29.6*
5	**48**	**50**	**300**	**500**	**0.5**	**0.5**	**NO**	167	1664	23.9	24.8	28.2

Tabla 10. Resultados BCI con 3PS

Se puede observar en este caso al igual que en el caso de la integración de dos perceptrones simples como también todas las pruebas (a pesar de realizar combinaciones de población, generaciones e iteraciones) devuelven resultados prácticamente iguales, de manera que al igual que en el apartado anterior no existe más de un 3% de diferencia entre el mejor resultado y el peor. El mejor resultado obtenido se ha conseguido con muy pocas generaciones y se corresponde con el de la prueba tres y que obtiene un 73.3% de aciertos en validación.

Al comparar las pruebas 2 y 3 vemos como a pesar de haber cambiado la razón de aprendizaje y las iteraciones los resultados también son prácticamente iguales, por lo que se puede considerar que en este caso no es importante la razón de aprendizaje siempre que se encuentre dentro de unos valores, ya que como vimos en las pruebas simples con una razón de aprendizaje de 0.1 el resultado era bastante peor.

Al realizar la integración de los tres perceptrones simples sobre el primer caso en el algoritmo genético multiobjetivo se ha conseguido una ligera mejora sobre el uso aislado de los tres perceptrones simples: del 73.3% de aciertos (26.7% de error) en lugar del 72.29% de acierto, aunque no se han conseguido mejorar en gran medida. Con ello, se puede concluir que lo importante en este dominio era el clasificador utilizado y el hecho de utilizar el algoritmo genético multiobjetivo con FLN podría mejorar los resultados en caso de que el clasificador fuera malo para el dominio (como ocurría con los dos perceptrones), pero en caso de que el resultado fuese bueno por sí solo, no se consigue mejorarlo en gran medida.

También nos vemos en la obligación de comparar estos resultados con los de la competición sobre el sujeto uno y que se encuentran en la siguiente página web http://ida.first.fraunhofer.de/projects/bci/competition_iii/results/index.html, la posición en este caso, a pesar de haber mejorado apenas un 2% el resultado subiría de la 15 a la 11, ya que como se dijo anteriormente todos los resultados están más o menos en los mismos valores, siendo además el mismo valor sobre el sujeto 1 que el obtenido en el ANEXO 1 "Estudio datos BCI".

6.4. Descripción de los datos de otros dominios

Se han estudiado también otros dominios aparte del BCI, que han sido típicamente utilizados para comparar diversos algoritmos de aprendizaje automático. En primer lugar se obtuvieron datos sobre el cáncer y la diabetes de la base de datos de Proben preparados por Prechelt [Prechelt94], de manera que se utilizarán los datos existentes en los ficheros cancer1.dt y diabetes1.dt para nuestras pruebas.

En lo referente al fichero del cáncer tiene nueve atributos con valores reales pero se encuentran codificados en valores entre 0 y 1. Además la clase está codificada en binario y será 2 en el caso de benigno y 4 en el caso de maligno. Existen 699 ejemplos, de los cuales 458 se trata de ejemplos benignos y 241 malignos. Los datos se dividirán en tres conjuntos, que se dividirán de la siguiente forma: 350 para entrenamiento, 175 para test y 174 para validación. El dominio en este caso solamente tiene un fichero cancer1.dt, aunque aparece en el fichero Readme.txt el número de ejemplos que deberá tener cada fichero, y además se siguió la separación realizada en el siguiente artículo [Sierra01] para poder comparar los resultados con los del artículo.

En lo referente al fichero de la diabetes tiene ocho atributos que se encuentran codificados en valores comprendidos entre 0 y 1, más el valor de la clase que será 0 si no tiene diabetes y 1 en caso contrario. El dominio en este caso solamente tiene un fichero diabetes1.dt, aunque aparece en el fichero Readme.txt el número de ejemplos que deberá tener cada fichero, y además se siguió la separación realizada en el siguiente artículo [Sierra01] para poder comparar los resultados con los del artículo.

La información de los atributos se puede ver en la tabla 11:

NÚMERO DE ATRIBUTO	INFORMACIÓN DEL ATRIBUTO
0	NÚMERO DE EMBARAZOS
1	CONCENTRACIÓN DE GLUCOSA EN SANGRE A LAS 2 HORAS EN UNA PRUEBA DE TOLERANCIA DE GLUCOSA ORAL
2	TENSIÓN ARTERIAL
3	GROSOR DE LA PIEL DEL TRÍCEPS
4	CANTIDAD DE INSULINA QUE TOMAN EN DOS HORAS
5	IMC(ÍNDICE DE MASA CORPORAL)
6	FUNCIÓN DE PEDIGRÍ DE DIABETES
7	EDAD

Tabla 11. Descripción de los atributos de diabetes

Existen un total de 768 ejemplos, de los cuales 500 se corresponden con ejemplos que no tienen diabetes y 268 que sí. Los ejemplos se dividirán en tres conjuntos de la siguiente forma: 384 para entrenamiento, 192 para test y 192 para validación.

También se ha estudiado otro dominio denominado cuadrático. La peculiaridad de este dominio es que es artificial y está generado de tal manera que la superficie de separación es una expresión lineal en

términos de atributos producto, por lo que nuestros experimentos deberían llegar a buenas soluciones. Además se pueden generar tantos datos como se deseen. El dominio cuenta con ocho atributos más la clase que será [-1,1]. Existirán tres conjuntos de datos con 700 ejemplos para entrenamiento, 300 para test y 1000 para validación.

6.5. Resultados otros dominios con dos perceptrones simples

Para estos dominios no se han tomado referencias de tiempos, ya que los resultados obtenidos eran casi instantáneos, el tiempo que tardaban se puede decir que era en todos los casos menor a un segundo, salvo para el dominio cuadrático que podría ser inferior a dos segundos.

Antes de mostrar los resultados para cada uno de los dominios, hay que especificar que los resultados vendrán dados en porcentaje de aciertos, así por ejemplo un resultado de 100, se trataría de 100% de aciertos, que significaría que se habrían acertado todos los ejemplos. En cuanto a las iteraciones se refiere al número de iteraciones que se ejecutarán antes de parar la ejecución.

Los resultados para el dominio del cáncer con los dos perceptrones simples combinados se pueden ver en la tabla 12:

ITERACIONES	TRAIN	TEST	VALIDACIÓN
300	**96.85**	**97.71**	**98.27**
500	**96.85**	**97.71**	**98.27**

Tabla 12. Cáncer: Resultado perceptron simple

Se puede observar como los resultados son muy buenos, ya que llegan a un porcentaje de aciertos muy grande, casi cercano al 100% y no se observa que haya sobreaprendizaje. Además como se ofrecen los mismos resultados utilizando 300 o 500 iteraciones, se utilizarán 300 iteraciones para realizar las pruebas posteriores.

Los resultados para el dominio de la diabetes se pueden ver en la tabla 13:

ITERACIONES	TRAIN	TEST	VALIDACIÓN
300	**76.82**	**75**	**77.08**
500	**77.08**	**76.04**	**72.91**
700	**77.08**	**76.04**	**72.91**

Tabla 13. Diabetes: Resultado perceptron simple

Con estos resultados se puede ver cómo a pesar de obtener un mejor resultado tanto en entrenamiento como en test utilizando más iteraciones, además el incremento en iteraciones prácticamente no mejora los resultados de test y por tanto se utilizarán en principio 300 iteraciones para los experimentos posteriores.

Los resultados para el dominio cuadrático se pueden ver en la tabla 14:

ITERACIONES	TRAIN	TEST	VALIDACIÓN
300	**66.14**	**62.66**	**62.59**
500	**66.28**	**64**	**64.59**
700	**66.28**	**64**	**64.59**

Tabla 14. Cuadrático: Resultado perceptron simple

A partir de estos resultados obtenidos con dos perceptrones simples, y con la ligera diferencia existente entre la utilización de 300 y 500 iteraciones y la gran capacidad de mejora que se presupone a los experimentos posteriores, se utilizarán en principio 300 iteraciones para no ralentizar los experimentos.

6.6. Resultados otros dominios con el software realizado

Para cada uno de los diferentes dominios se han realizado diferentes pruebas y para cada uno de ellos se ofrecen unos resultados distintos, por tanto vamos a tratar cada dominio de una manera particular.

Comúnmente a todos estos dominios se utilizaron como iteraciones del perceptron, una cuyas iteraciones aumentaban linealmente desde N/2 a N como se explicó en el apartado 5.6 "Algunas consideraciones adicionales", de manera que al principio de la ejecución el perceptron se ejecutaría menos veces e iría aumentando conforme aumentarán las generaciones.

6.6.1. Cáncer

Para el dominio del cáncer se parten de unos resultados de 96.85, 97.71 y 98.27 en porcentaje de aciertos para los conjuntos de entrenamiento, test y validación con el perceptron simple. Las pruebas realizadas no tardaron un tiempo superior a la media hora, de manera que se puede decir que su coste en tiempo no es muy grande en comparación con el tiempo que tardaban los experimentos de BCI.

6.6.1.1.Prueba 1

La primera prueba realizada utiliza una población de 48 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. Como se puede ver en la figura 63, que representa la evolución del frente cada 10 generaciones y donde la escala ha tenido que ser reducida debido a que los valores del porcentaje de error, que se encuentra en el eje vertical no subía del 5%. Como se puede ver en la figura el frente evoluciona de una manera razonable, pero se puede suponer que todavía podría evolucionar más, por lo que se realizarán pruebas con más generaciones y podrían existir más puntos por lo que se realizarán también más pruebas con un número mayor de individuos.

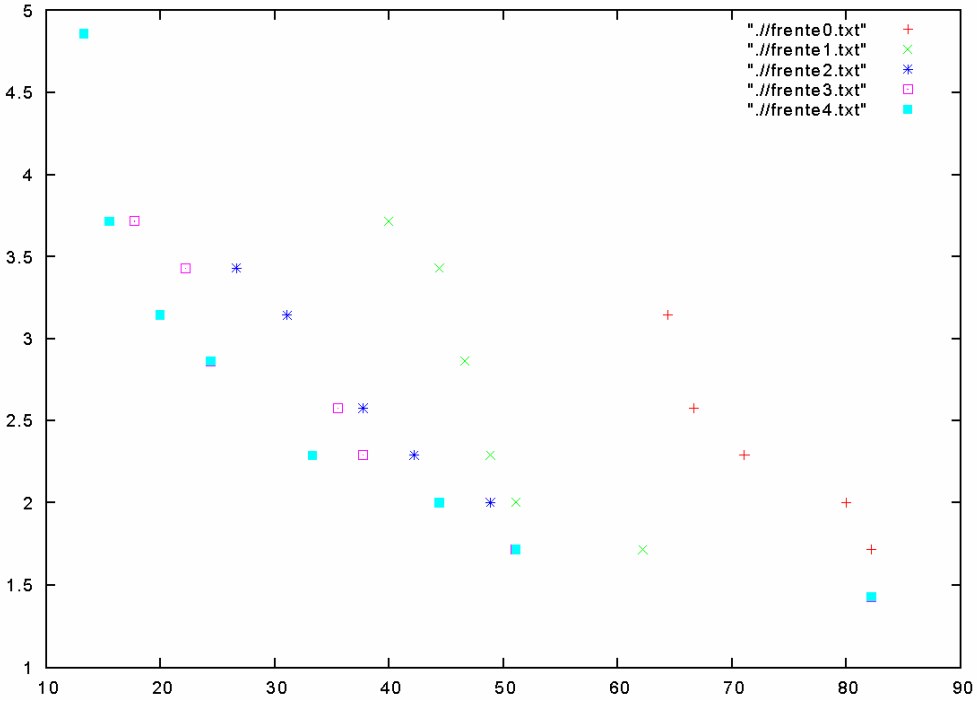


Ilustración 63. CÁNCER: Evolución frente prueba 1

En lo referente al frente de la última generación se puede ver en la figura 64 y está representado con las cruces verdes, se puede ver como los valores de test y validación no varían demasiado de los valores de entrenamiento. Se puede observar en especial como para un valor cercano el 42% de atributos, que sería el mejor punto del frente según se explicó en el 5.5 "Elección del punto del frente", el valor en test del porcentaje de error supera el 1%, lo cual sería aproximadamente un 98.9% en porcentaje de aciertos y como para la validación sería también un resultado aproximado de un 98.4% de aciertos.

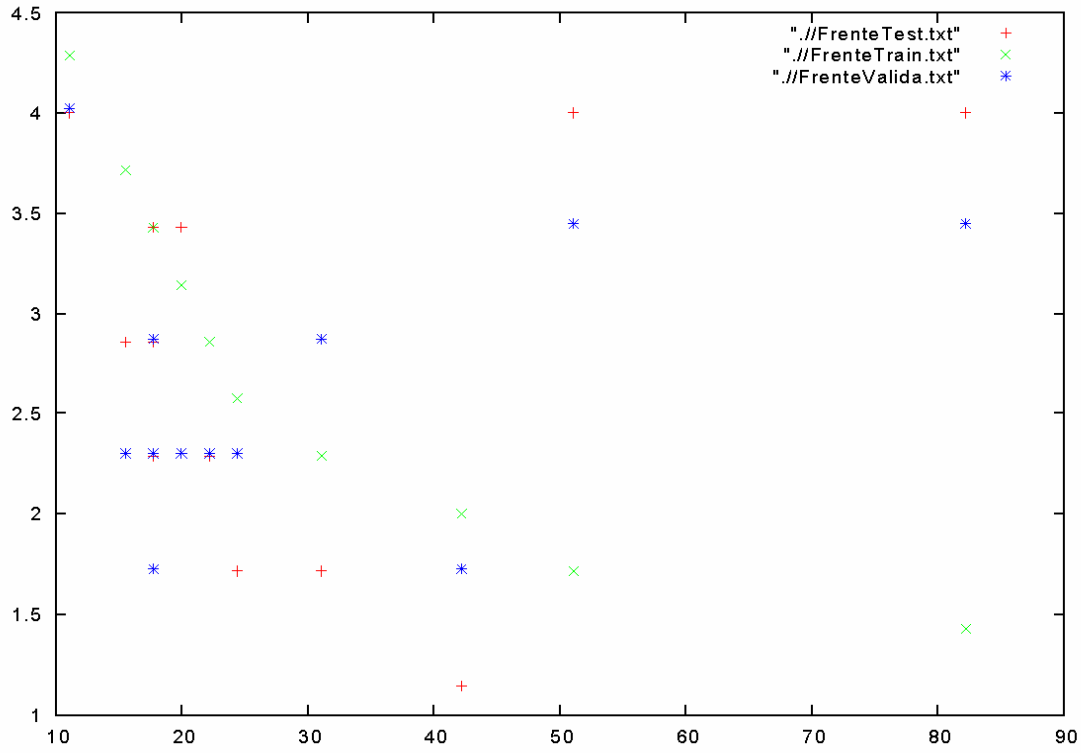


Ilustración 64. CÁNCER: Frente final prueba 1

6.6.1.2.Prueba 2

En la segunda prueba realizada se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 100, utilizando como número máximo de iteraciones 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En la figura 65 se representa la evolución del frente cada 10 generaciones y como se puede observar el frente varía desplazándose hacia la izquierda, de manera que se van encontrando soluciones mejores.

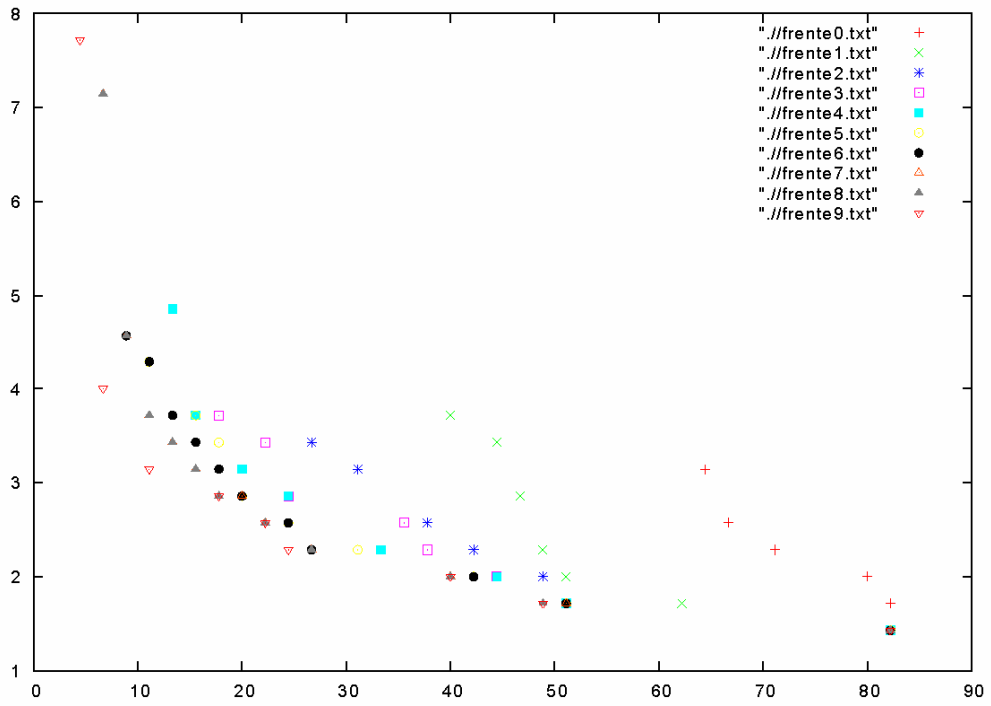


Ilustración 65. CÁNCER: Evolución frente prueba 2

Como se puede ver en la figura 66 los resultados son también bastante buenos y no existe prácticamente diferencia entre los resultados de entrenamiento y test y validación. Aunque no se aprecia bien en la figura el mejor punto en este frente de acuerdo con el conjunto de test se encuentra aproximadamente sobre el 25% de atributos y el 1.1% de fallos tanto en test como en validación. Por tanto, el resultado ha mejorado ligeramente el resultado en validación de la prueba anterior.

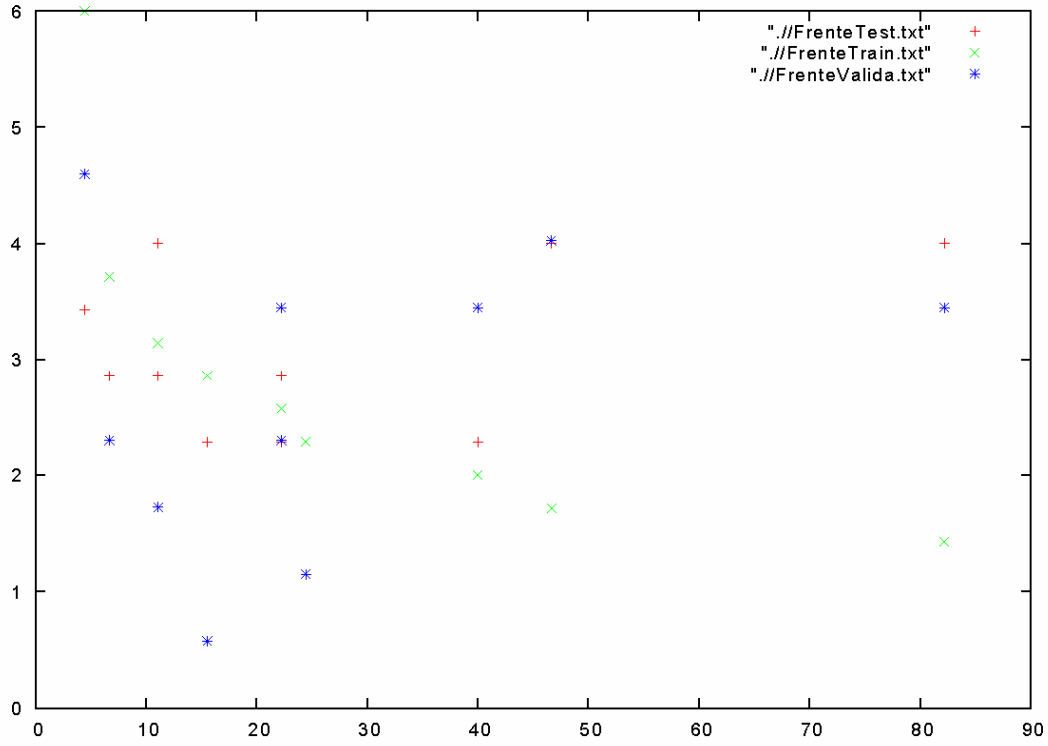


Ilustración 66. CÁNCER: Frente final prueba 2

6.6.1.3.Prueba 3

Para la tercera prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. Se puede observar como la evolución de los frentes cada 10 generaciones hace variar el frente hacia la izquierda, aunque no hay gran diferencia entre el frente mostrado en la figura 67 y el mostrado en la figura 63 de la prueba 1.

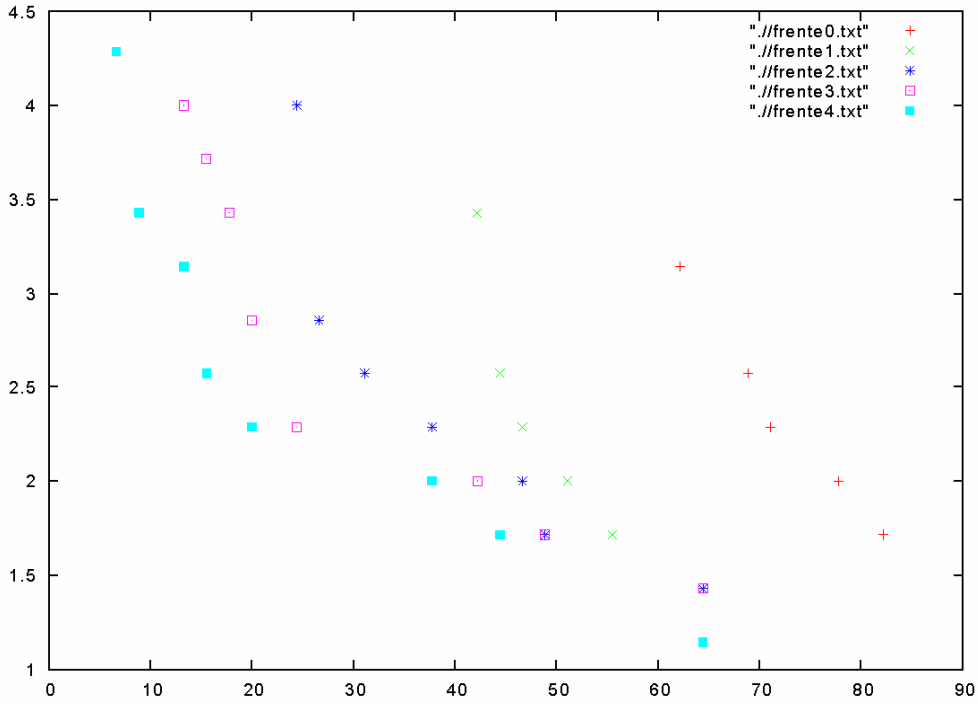


Ilustración 67. CÁNCER: Evolución frente prueba 3

En este prueba como se puede apreciar en la figura 68, los resultados no mejoran las pruebas anteriores, encontrándose el mejor punto de test aproximadamente sobre el 1.7 en porcentaje de fallos y sobre el 18% de atributos.

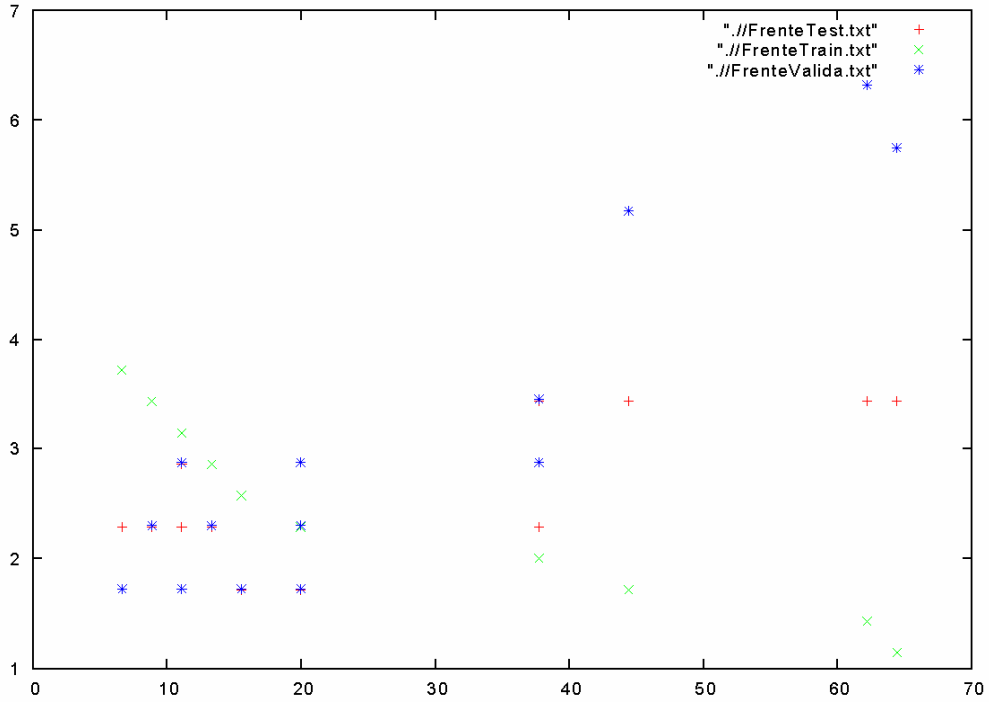


Ilustración 68. CÁNCER: Frente final prueba 3

6.6.1.4.Prueba 4

Para la cuarta prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 500. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. La diferencia entre esta prueba y la anterior estriba en el número de iteraciones máxima usada, y si comparamos la evolución de un frente con el anterior, podemos ver como en este caso se puede observar en la figura 69 como los frentes se mueven más rápido hacia la izquierda, es decir, que el aprendizaje sobre el conjunto de entrenamiento se realiza más rápido que en la prueba anterior.

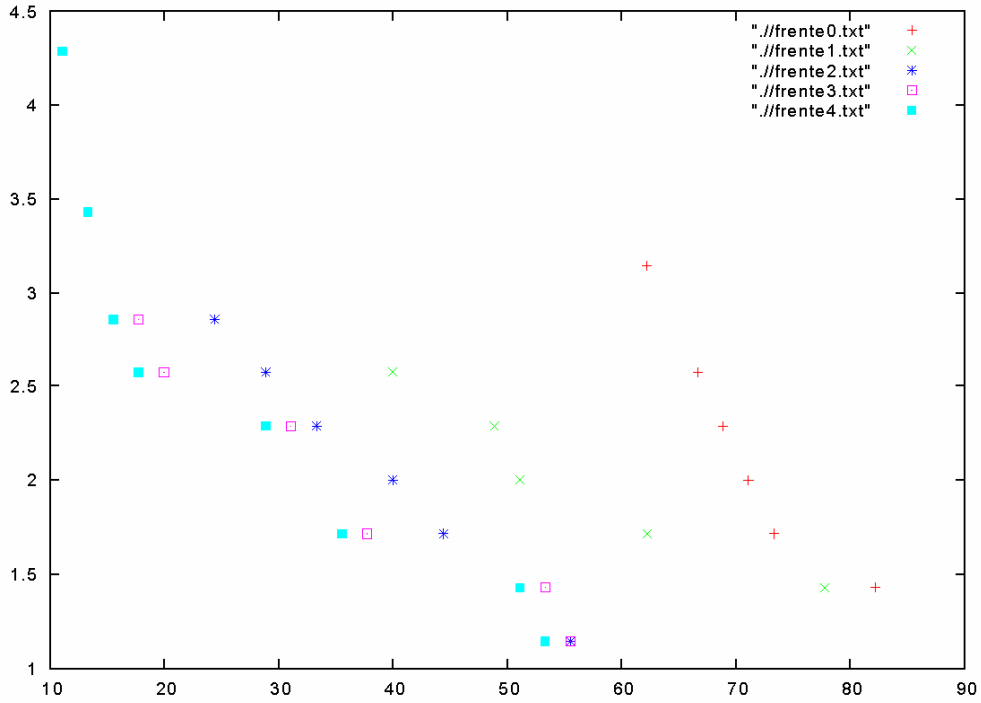


Ilustración 69. CÁNCER: Evolución frente prueba 4

Sin embargo, cuando vemos los resultados en la figura 70 sobre el frente final de entrenamiento, test y validación vemos como en este caso existen más puntos de test y validación que sobrepasan al resultado de entrenamiento. Por tanto, no parece adecuado utilizar más iteraciones porque los resultados parecen a simple vista mejores, aunque si realizamos una observación más profunda y cogemos el mejor punto de test, que se encuentra también sobre el 18% de atributos y tiene un porcentaje de error de aproximadamente el 1.1%, y cuyo resultado es parecido en validación.

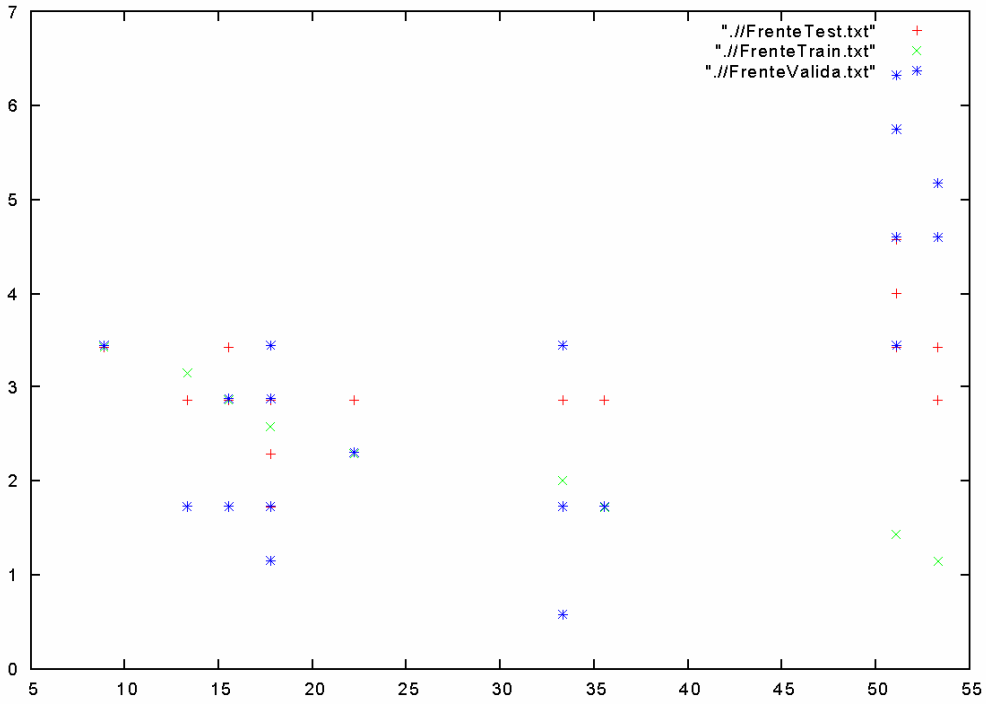


Ilustración 70. CÁNCER: Frente final prueba 4

6.6.1.5.Prueba 5

Para la quinta prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En esta prueba la evolución de los frentes nos ha llevado hasta un punto del frente en el que el individuo está compuesto de todo ceros, y por tanto su porcentaje de fallos es 100. Por tanto, además existe otro punto con muy pocos atributos, probablemente 1 y cuyo porcentaje de fallos está más o menos en el 65% de fallos.

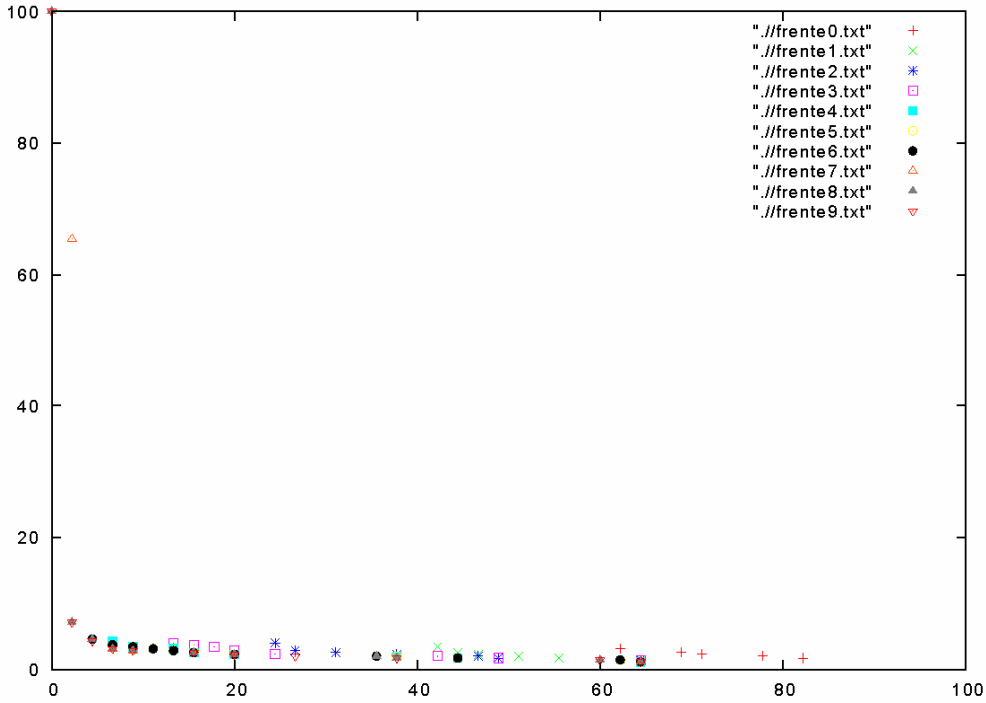


Ilustración 71. CÁNCER: Evolución frente prueba 5

Aunque en la figura 72 no se aprecien bien los datos, podemos decir que aunque el frente tenga más puntos los resultados obtenidos tomando el mejor punto de test son parecidos a las pruebas anteriores.

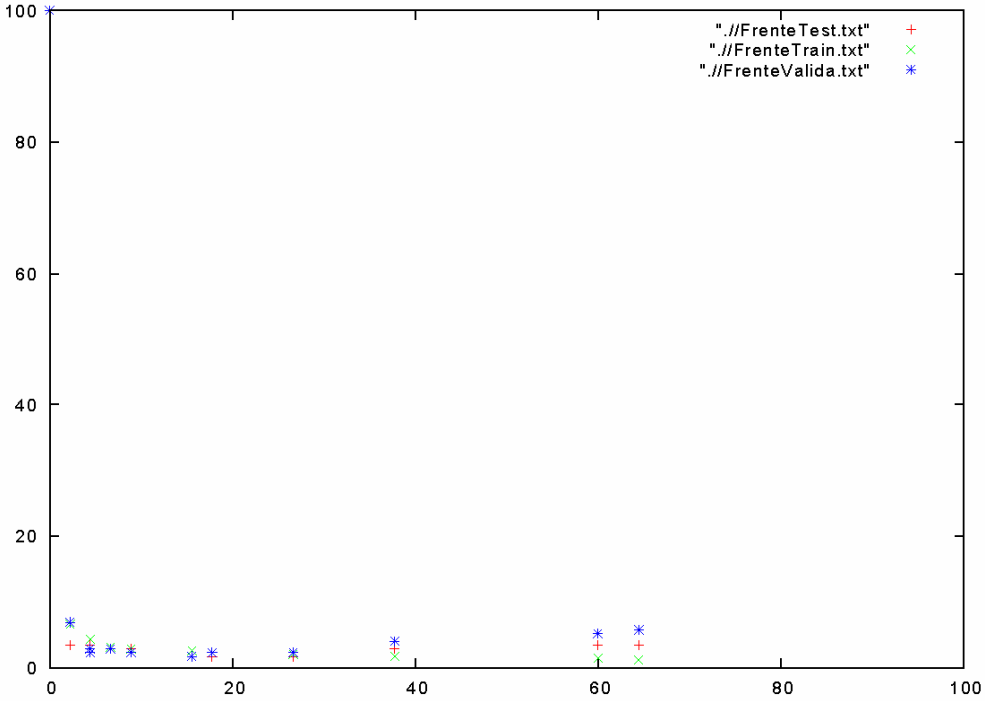


Ilustración 72. CÁNCER: Frente final prueba 5

6.6.1.6.Prueba 6

Para la sexta prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Al utilizar un nuevo objetivo se puede ver en la figura 73 como los ejes representan el porcentaje de fallos para cada una de las clases y como la evolución varía de forma diferente, también se observa que en el último frente pintado sólo existen dos puntos.

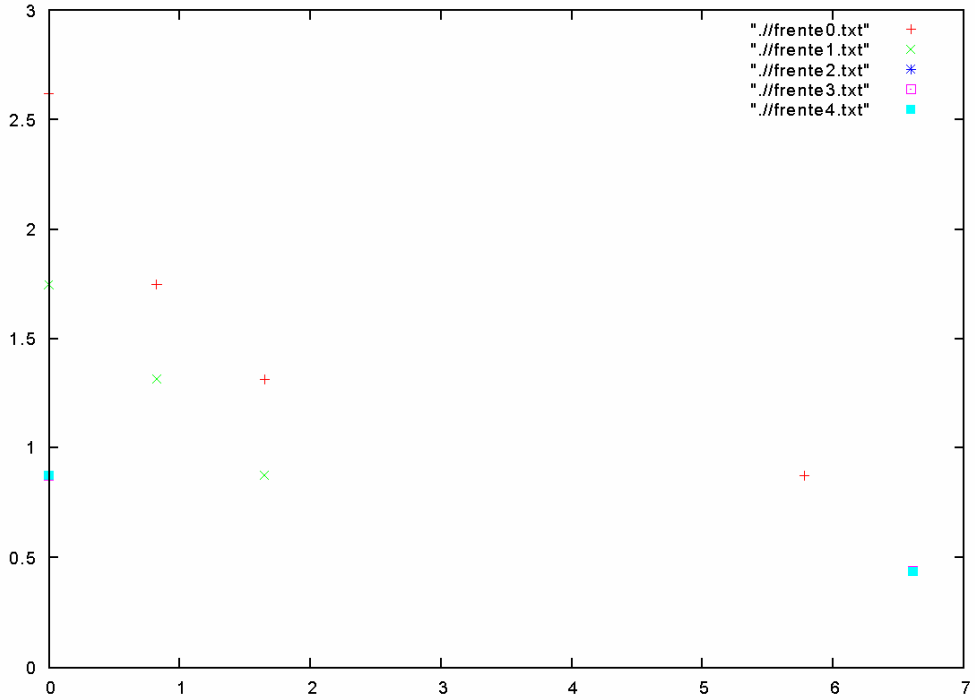


Ilustración 73. CÁNCER: Evolución frente prueba 6

Como ya se pudo observar en la evolución, efectivamente en la figura 74 se observa como el frente tiene solamente dos puntos y también como esos puntos, aunque en entrenamiento sean mejores que las pruebas anteriores, para test y validación sin embargo no llegan a los mismos resultados.

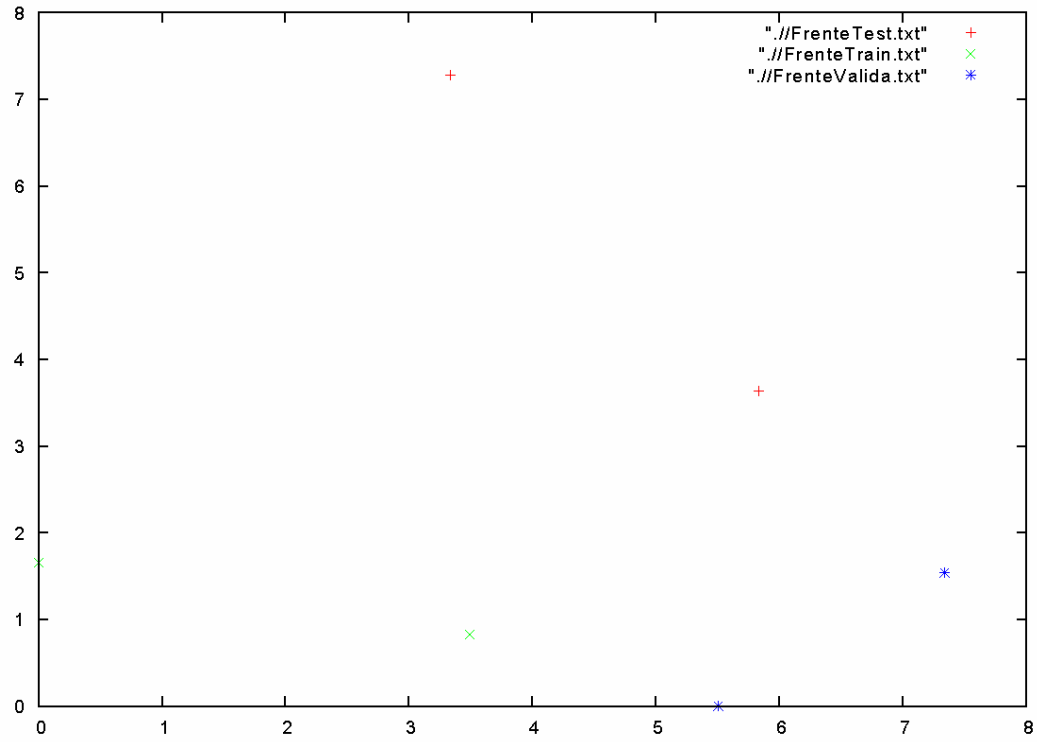


Ilustración 74. CÁNCER: Frente final prueba 6

6.6.1.7.Prueba 7

Para la séptima prueba se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Al realizar la prueba sobre más generaciones se puede observar en la figura 75 como los frentes tienen más puntos, aunque al final parece también se quedan en muy pocos.

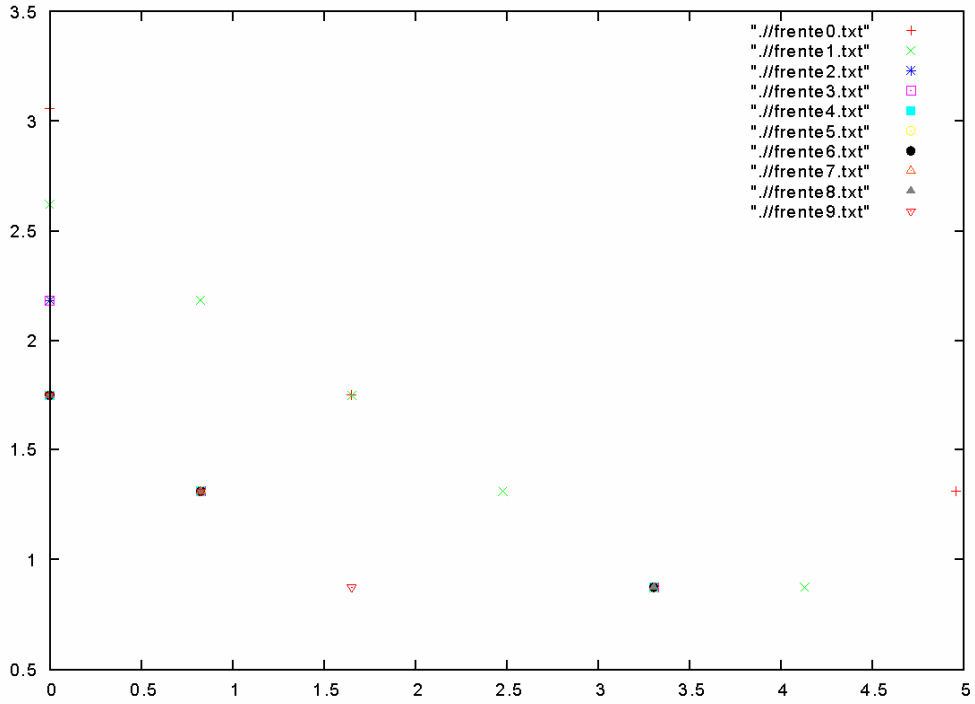


Ilustración 75. CÁNCER: Evolución frente prueba 7

En esta prueba se puede observar en la figura 76 como el frente parece más homogéneo, y aunque el resultado en entrenamiento sea algo peor que la prueba anterior, los resultados para test y validación son mejores, aunque sin llegar a los resultados ofrecidos cuando las pruebas se han realizado utilizando el caso 1.

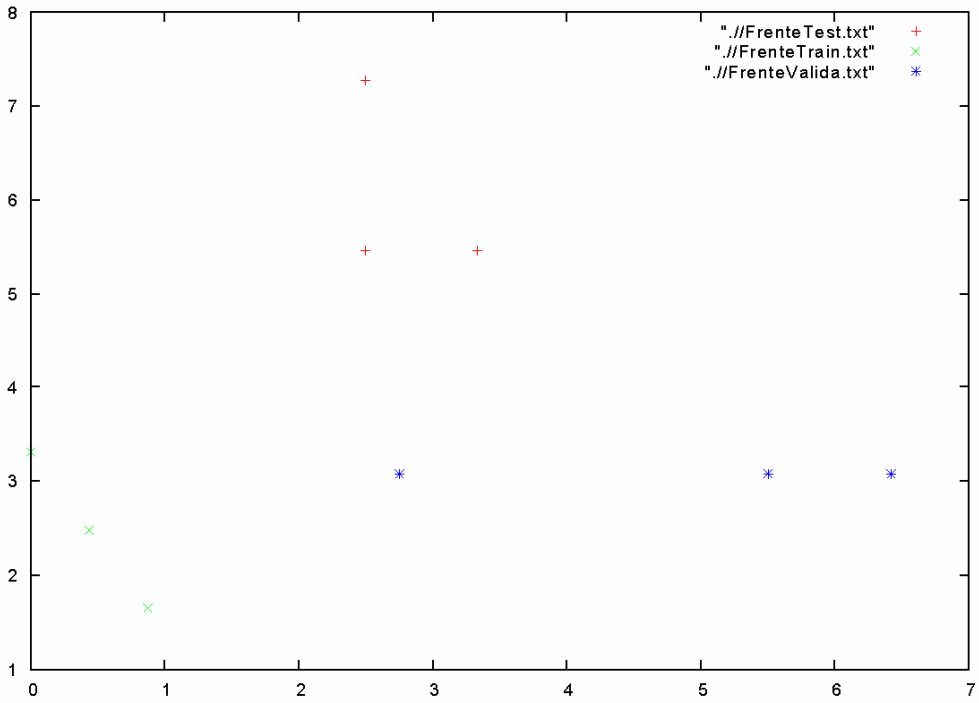


Ilustración 76. CÁNCER: Frente final prueba 7

6.6.1.8.Resultados finales

A continuación aparecen en la tabla 15 los resultados para cada una de las pruebas realizadas, en el que se apuntan sus resultados tomados del frente final y cogiendo como valor del frente el mejor punto en el conjunto de test. Para decidir cuál era el mejor punto, en el caso del objetivo uno simplemente se miraba el porcentaje de fallos total y se tomaba el punto con un menor porcentaje, mientras que para el objetivo dos se cogía el número de fallos de ambos objetivos, se realizaba la suma y se tomaba el valor con un menor número de fallos.

En lo referente a las pruebas con el caso 1, se puede ver como los resultados son bastante parecidos en todas las pruebas, aunque lo que varía es el número de atributos en la mejor solución que se puede ver en el objetivo cero.

Si comparamos los resultados obtenidos utilizando el caso uno y el dos, se puede ver como aunque para el caso dos se obtienen mejores resultados en entrenamiento, luego en test y validación los resultados son bastante peores, además se podía observar también en los frentes como cuando se utilizaba el caso dos estos frentes tenían muy pocos puntos. En la tabla 16 se muestran los resultados para el cáncer.

Caso	**Pob.**	**Gen.**	**Ite.**	**Obj0**	**Obj1**	**Total**	**%Tot**	**Test**	**%Test**	**Valid.**	***%Valida.***
1	**48**	**50**	**300**	19	7	7	2	2	1,14	3	*1,72*
1	**48**	**100**	**300**	11	8	8	2,28	2	1,14	2	*1,15*
1	**100**	**50**	**300**	7	9	9	2,57	3	1,71	3	*1,72*
1	**100**	**100**	**300**	7	9	9	2,57	3	1,71	3	*1,72*
1	**100**	**50**	**500**	8	9	9	2,57	3	1,71	2	*1,15*
2	**100**	**50**	**300**	0	2	2	0,57	8	4,57	9	*5,17*
2	**48**	**100**	**300**	1	3	4	1,14	6	3,42	5	2,87

Tabla 15. Resultados Cáncer

Si tomamos el mejor de todos los obtenidos en test, las pruebas una y dos serían las elegidas, con valores de 1.15 y 1.72 para validación. Ahora vamos a comparar nuestros resultados con otros de la literatura que también utilizaron functional link [Sierra01] y que para un grado polinomial de dos, como el utilizado en nuestras pruebas sus resultados son 1.57 para validación, con lo que se puede decir en esta prueba que los resultados son similares. Si los comparamos con el que de grado polinomial tres, es decir 1.84 también son similares.

6.6.2. Diabetes

Para el dominio de la diabetes partimos de unos resultados con el perceptron simple de 76.82, 75 y 77.08 respectivamente para los conjuntos de entrenamiento, test y validación. Para este dominio las pruebas no eran muy costosas, ya que no llegaron a superar la hora de ejecución ninguna de ellas.

6.6.2.1.Prueba 1

Para la primera prueba se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En la figura 77 se puede observar como el frente va moviéndose hacia la izquierda de manera que utiliza menos atributos llegando a los mismos resultados, ya que no consigue bajar de más o menos el 20% de tasa de error total.

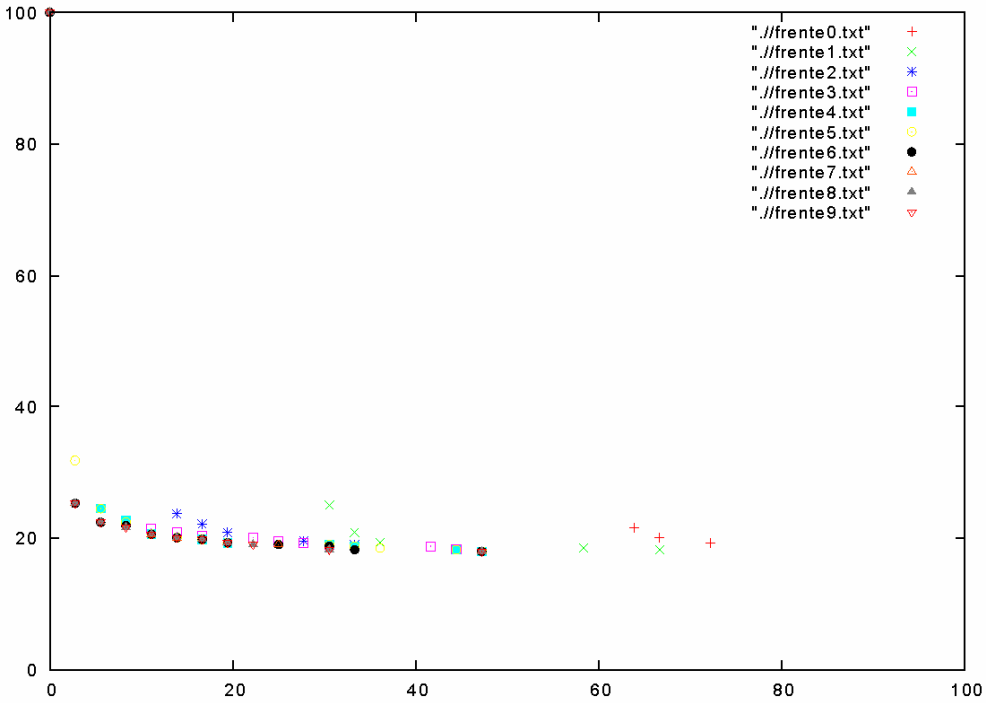


Ilustración 77. DIABETES: Evolución frente prueba 1

En la figura 78 se puede observar como los resultados se mantienen cerca del 20% de fallos y como los resultados para test y validación se mantienen parecidos a los de entrenamiento, aunque ligeramente superiores. El mejor punto para test se encuentra más o menos en el 33% de atributos escogidos y tiene más o menos un 20% de fallos en test, siendo en validación un poco mayor ese porcentaje.

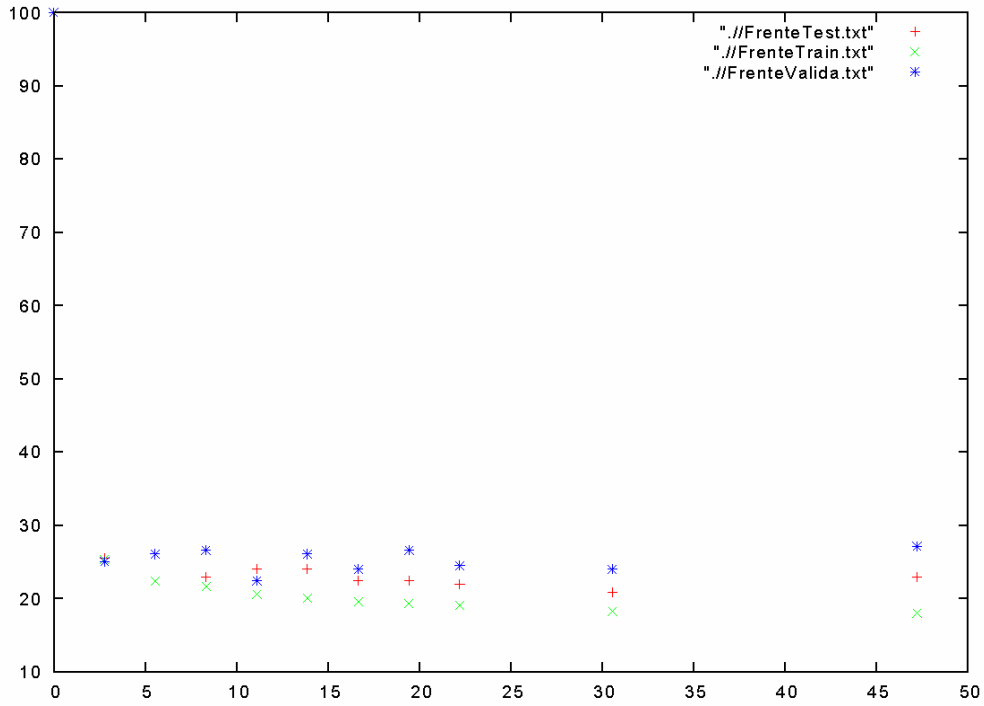


Ilustración 78. DIABETES: Frente final prueba 1

6.6.2.2.Prueba 2

Para la segunda prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En este caso se puede observar en la figura 79 como los frentes evolucionan más rápido al principio debido a la mayor cantidad de población y como al final se ralentiza este aprendizaje.

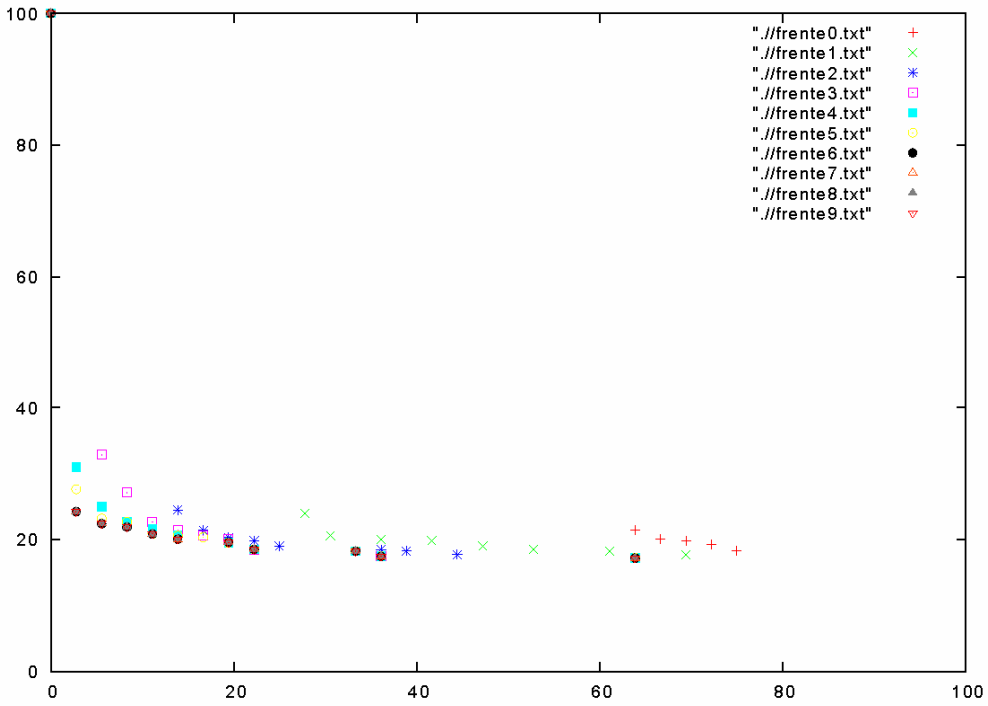


Ilustración 79. DIABETES: Evolución frente prueba 2

Se puede observar como estos resultados parecen ligeramente mejores en entrenamiento que los de la prueba anterior, aunque para validación parece que varían un poco más y en porcentaje de error que los anteriores. El mejor punto del frente para el test se encuentra sobre el 20% de atributos escogidos y es algo peor tanto en test como en validación que el anterior.

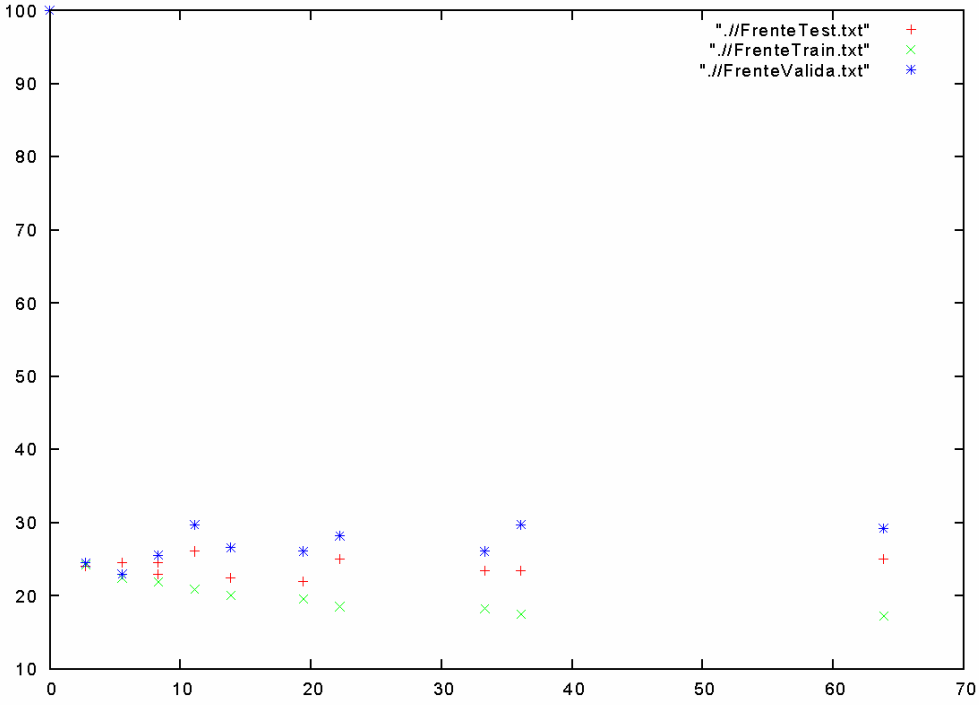


Ilustración 80. DIABETES: Frente final prueba 2

6.6.2.3.Prueba 3

Para la tercera prueba se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 500. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En esta prueba se decidió recurrir a utilizar más iteraciones para descubrir si se podía llegar a un mejor resultado. El frente evoluciona de manera parecida a los anteriores, como se puede ver en la figura 81.

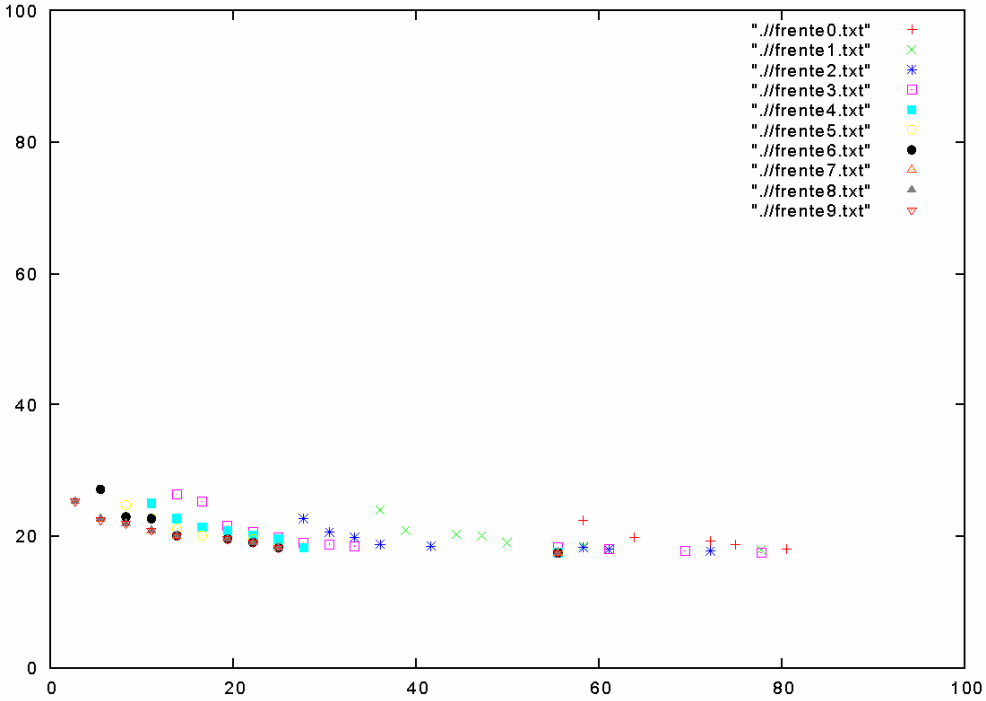


Ilustración 81. DIABETES: Evolución frente prueba 3

En este caso el frente es bastante parecido a los dos anteriores, aunque parece que existe más separación entre los frentes de test y validación con el de entrenamiento. Aunque los resultados son peores, teniendo el punto de test un porcentaje del 22%.

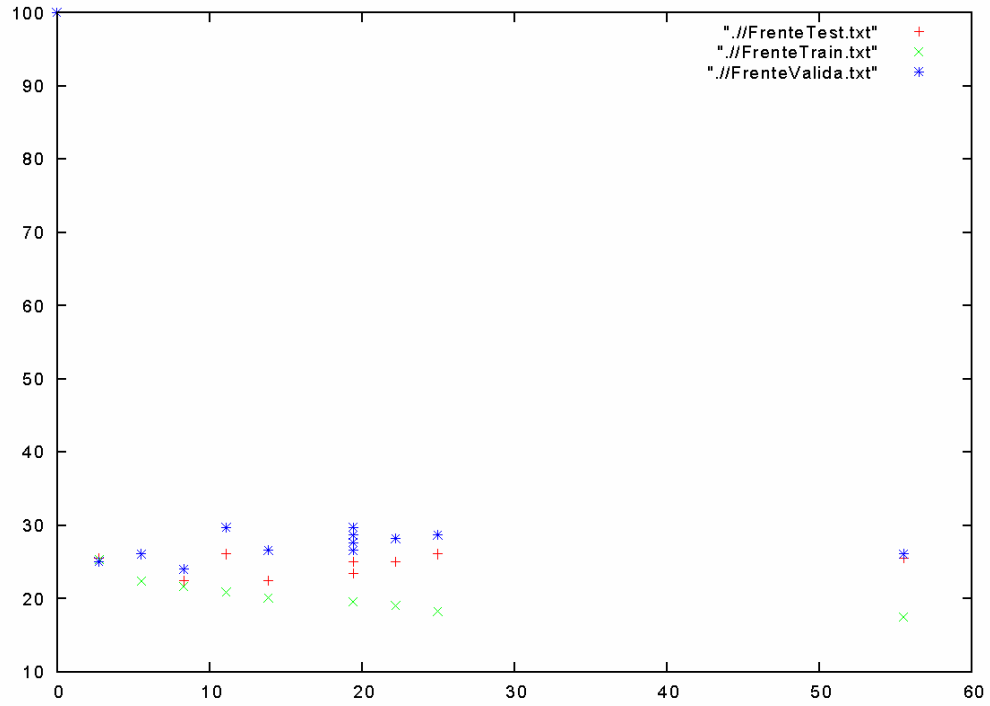


Ilustración 82. DIABETES: Frente final prueba 3

6.6.2.4.Prueba 4

Para la cuarta prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 500. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En la figura 83 se puede observar como la evolución de los frentes es similar a todos los anteriores experimentos.

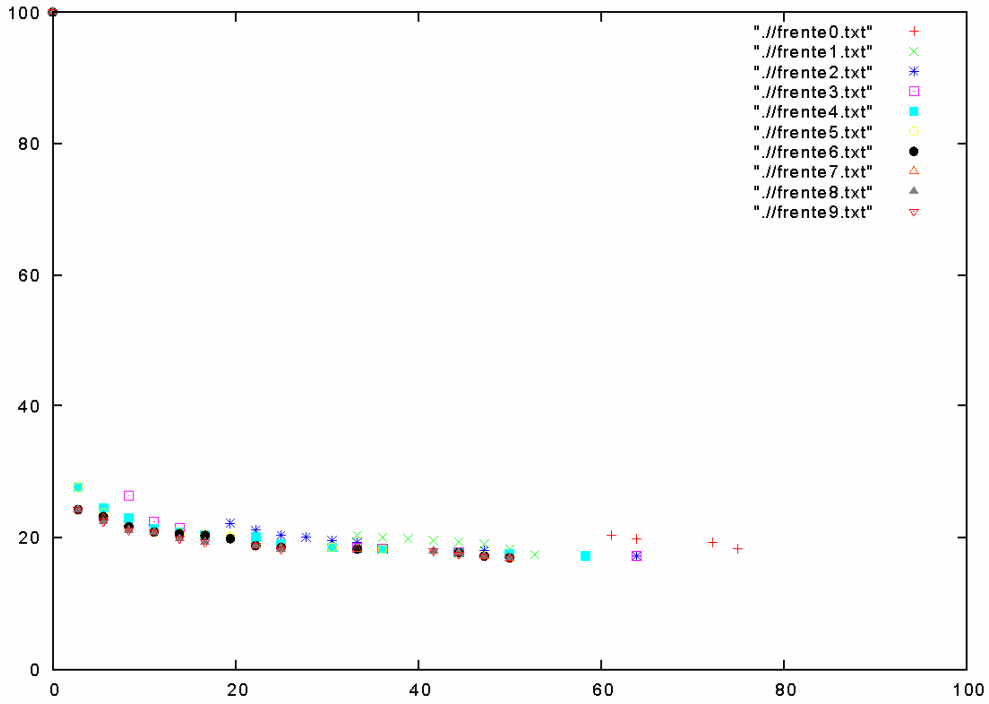


Ilustración 83. DIABETES: Evolución frente prueba 4

En los frentes de la figura 84 se puede observar como el frente de validación se ha acercado al de test con respecto a la prueba anterior, y como aunque el mejor punto en test tiene un 14% de atributos, y aunque sigue estando en un 22%, para validación ha bajado hasta el 23.5%.

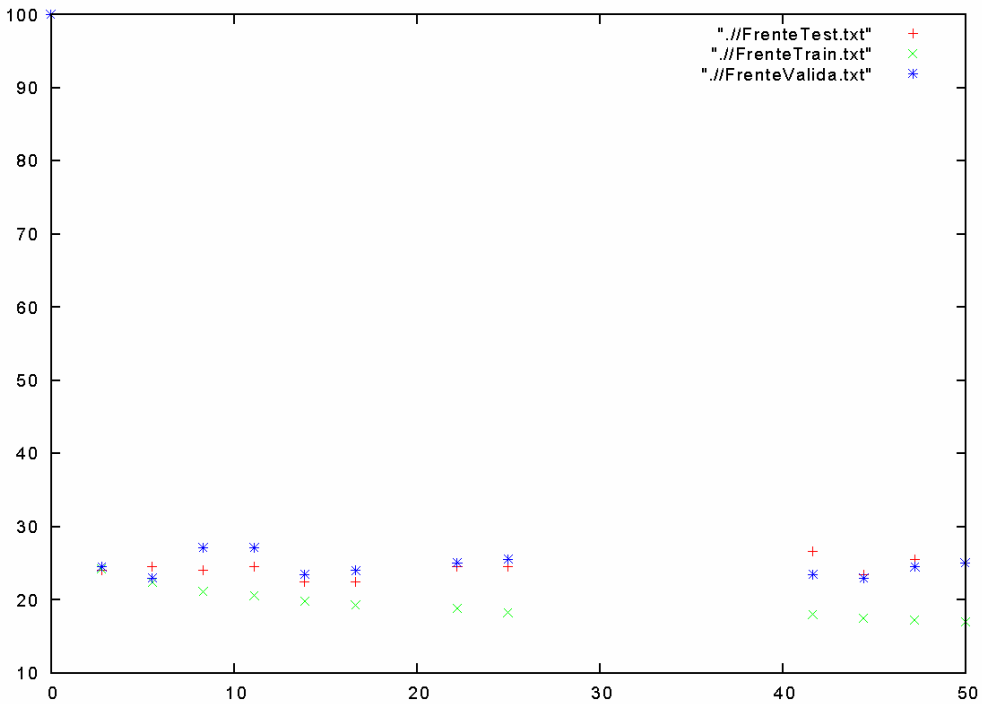


Ilustración 84. DIABETES: Frente final prueba 4

6.6.2.5.Prueba 5

Para la quinta prueba se utilizó una población de 200 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 500. También cabe resaltar que la prueba se realizó sobre el primer caso, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. Se puede observar con esta prueba como el frente ya había dejado de aprender mucho antes de terminar la ejecución, puesto que a partir del frente 10 no existe casi variación, y como a partir del 15 todos los frentes son prácticamente iguales.

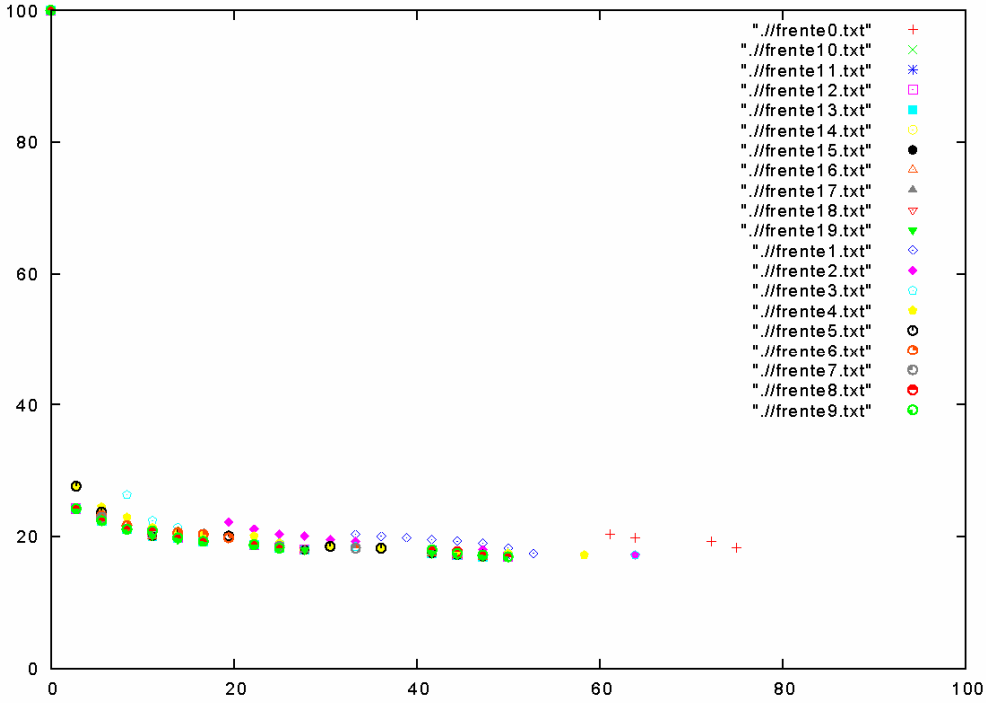


Ilustración 85. DIABETES: Evolución frente prueba 5

Observando estos frentes de la figura 86 y comparándolos con las pruebas anteriores, podemos llegar a la conclusión que utilizando estos parámetros no se llegará a una mejor solución, por lo que aumentar el número de generaciones, la población o las iteraciones no llevarán a una mejor solución. El mejor punto de este frente para test, llega a los mismos resultados, tanto para test como para validación que los mostrados en la prueba 3.

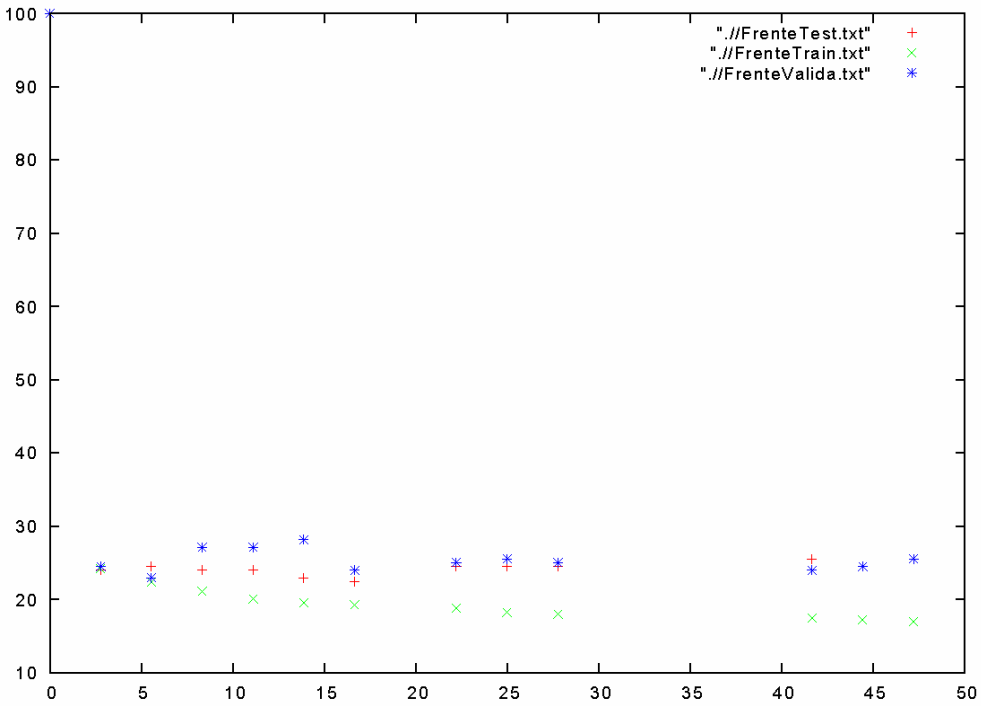


Ilustración 86. DIABETES: Frente final prueba 5

6.6.2.6.Prueba 6

Para la sexta prueba se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Al utilizar un nuevo objetivo se puede ver en la figura 87 como los ejes representan el porcentaje de fallos para cada una de las clases y como la evolución varía de forma diferente, también se como es la clase 1, representada en el eje vertical, la más difícil de aprender, ya que mientras que la clase dos como máximo falla un 20%, para la clase 1 este valor puede llegar al 60%.

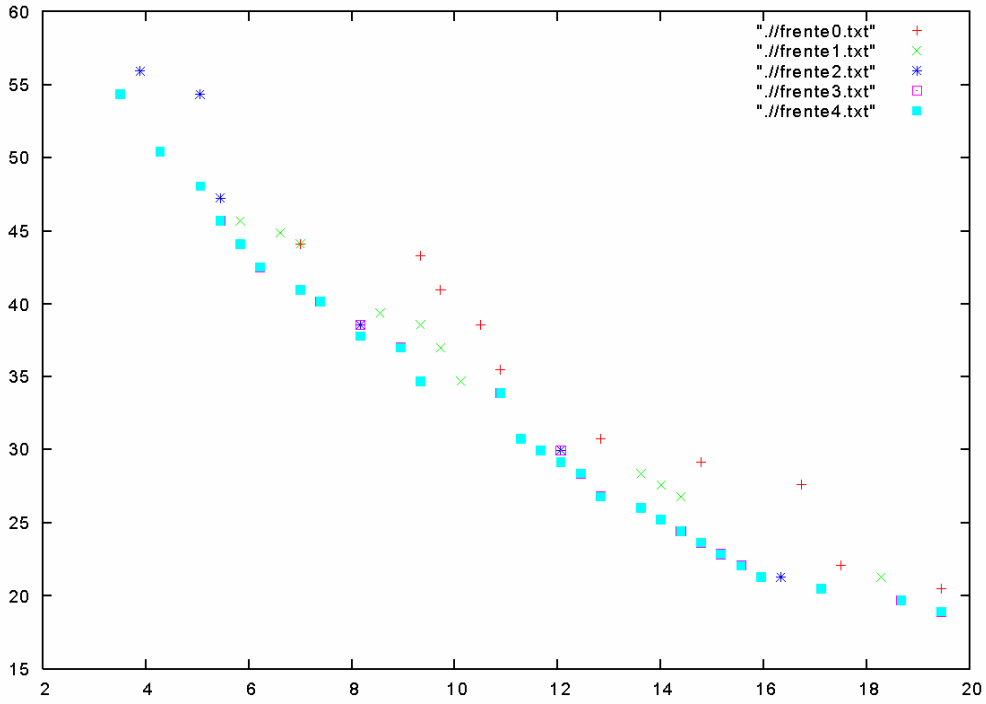


Ilustración 87. DIABETES: Evolución frente prueba 6

Se puede observar en la figura 88 como en este caso el frente final tiene más puntos que los frentes realizados con el caso 1, esto no ocurría así para el caso del cáncer, también podemos darnos cuenta de que no podemos saber cuál es el punto de test y validación que se corresponden con el de entrenamiento, ya que podría ser cualquiera de ellos debido a que en este caso los frentes tienen muchos puntos y además en los ejes se representan los porcentajes de fallos para cada clase, por lo que no coinciden los puntos de entrenamiento, test y validación en un eje como ocurría con las pruebas del caso 1, aunque cabe pensar que estarán próximos al punto del frente en entrenamiento. Los resultados sin embargo, no son mejores que los conseguidos en las pruebas anteriores.

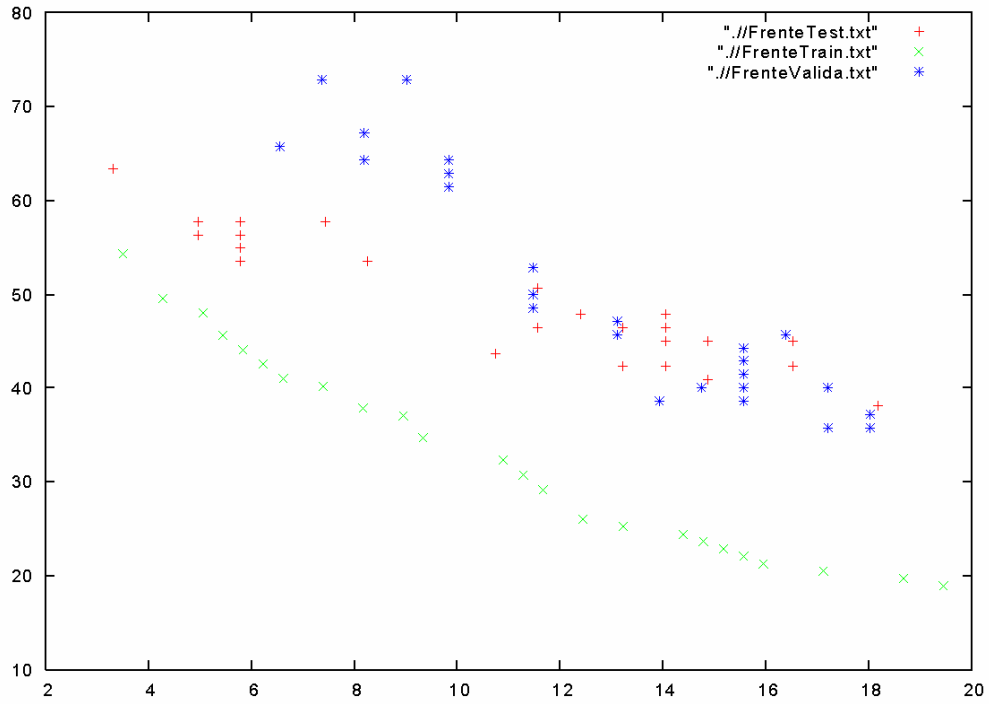


Ilustración 88. DIABETES: Frente final prueba 6

6.6.2.7.Prueba 7

Para la séptima prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. En la figura 89 se puede observar como los frentes van evolucionando y se puede ver como incluso en el último frente siguen apareciendo más puntos.

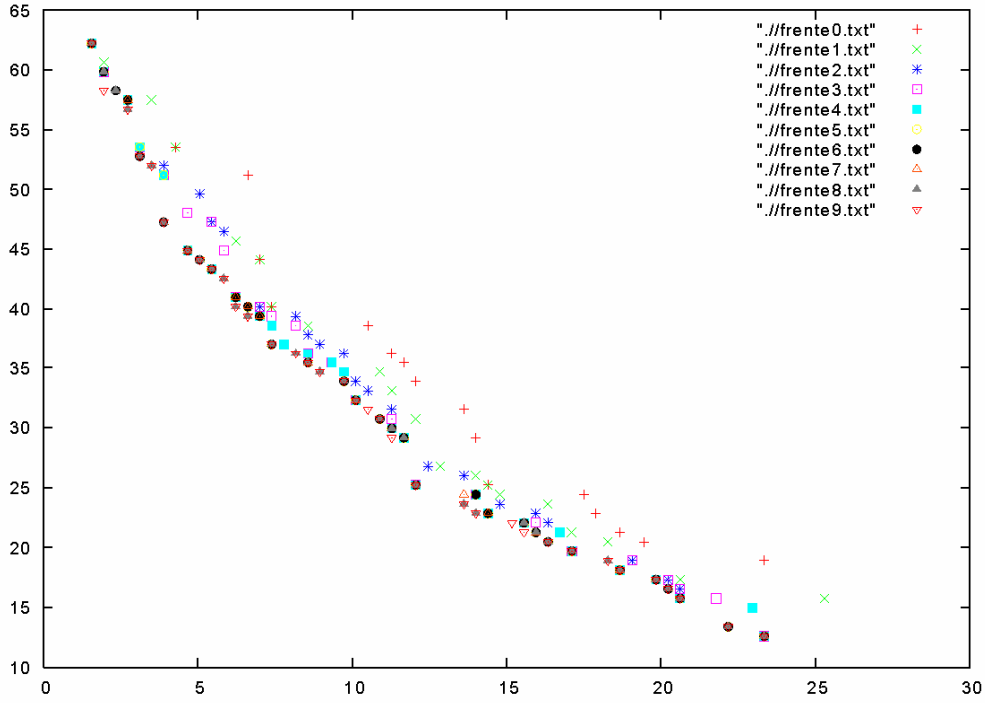


Ilustración 89. DIABETES: Evolución frente prueba 7

Lo que se puede sacar como conclusión al observar la figura 90 es que el frente tiene muchos puntos y se encuentra distribuidas con muchos puntos en todas sus partes, es decir, que se encuentran soluciones deseadas. Lo único que los resultados en este caso, tampoco son buenos.

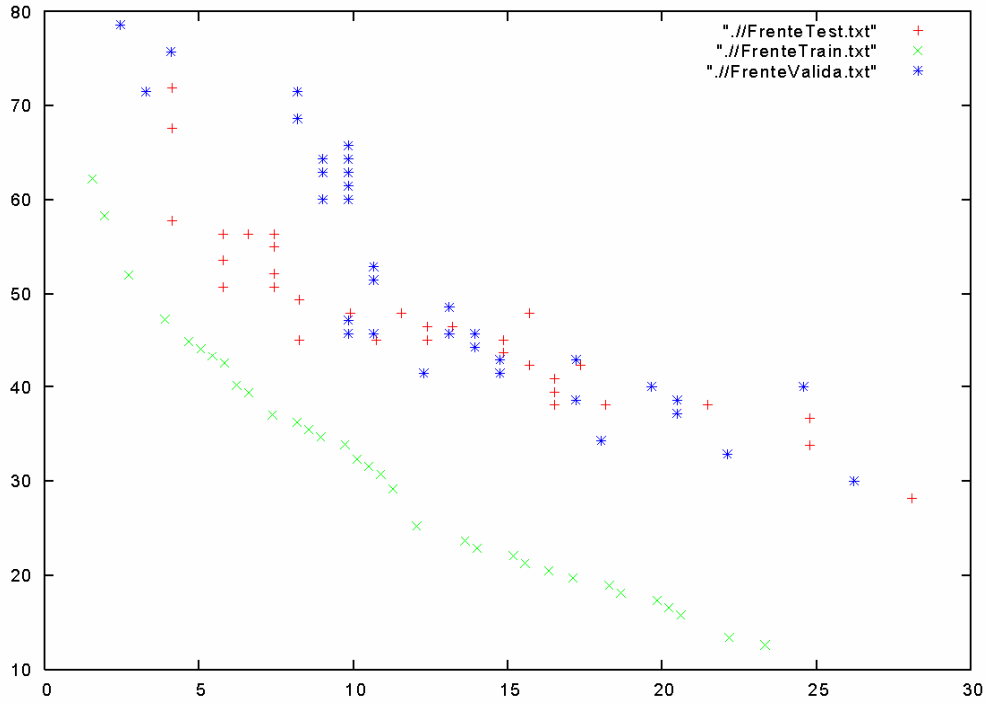


Ilustración 90. DIABETES: Frente final prueba 7

6.6.2.8.Prueba 8

Para la octava prueba se utilizó una población de 200 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Se ha tratado de utilizar más individuos como población, con el fin de intentar obtener más puntos del frente, con el fin de poder encontrar alguno que mejore los resultados encontrados anteriormente.

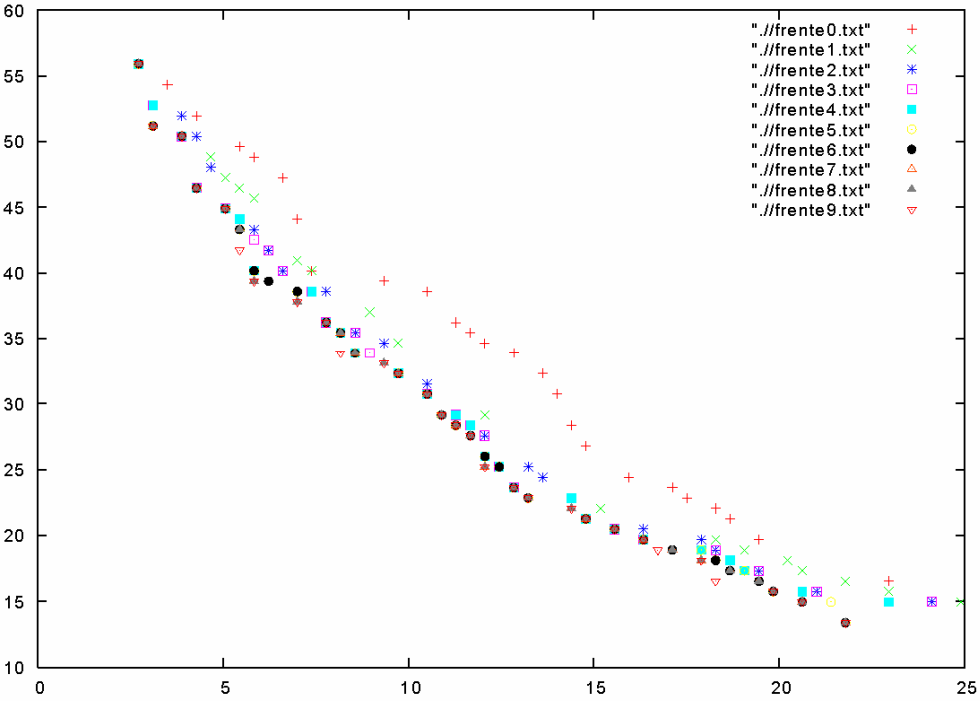


Ilustración 91. DIABETES: Evolución frente prueba 8

Aunque no se pueda apreciar en la figura 92, en este frente se ha obtenido un mejor resultado que con las dos anteriores, aunque sigue sin ser el mejor de los encontrados hasta el momento, por lo que habrá que intentar modificar otra variable para intentar obtener un mejor resultado.

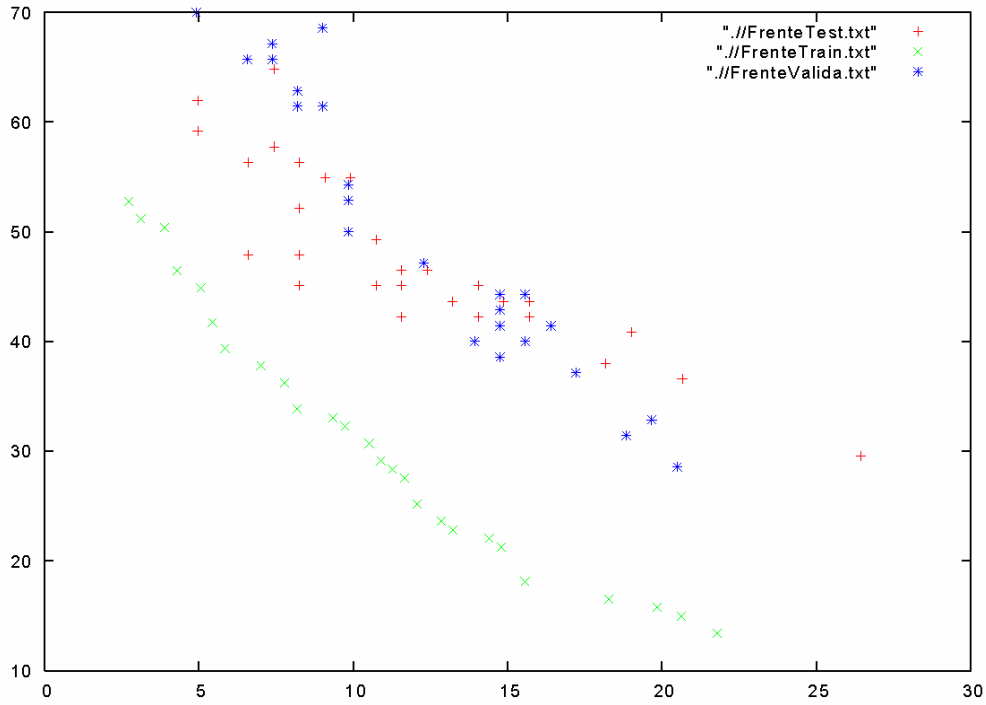


Ilustración 92. DIABETES: Frente final prueba 8

6.6.2.9.Prueba 9

Para la novena prueba se utilizó una población de 100 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 500. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases. Se decidió modificar el número de iteraciones máxima con la intención de encontrar un mejor resultado.

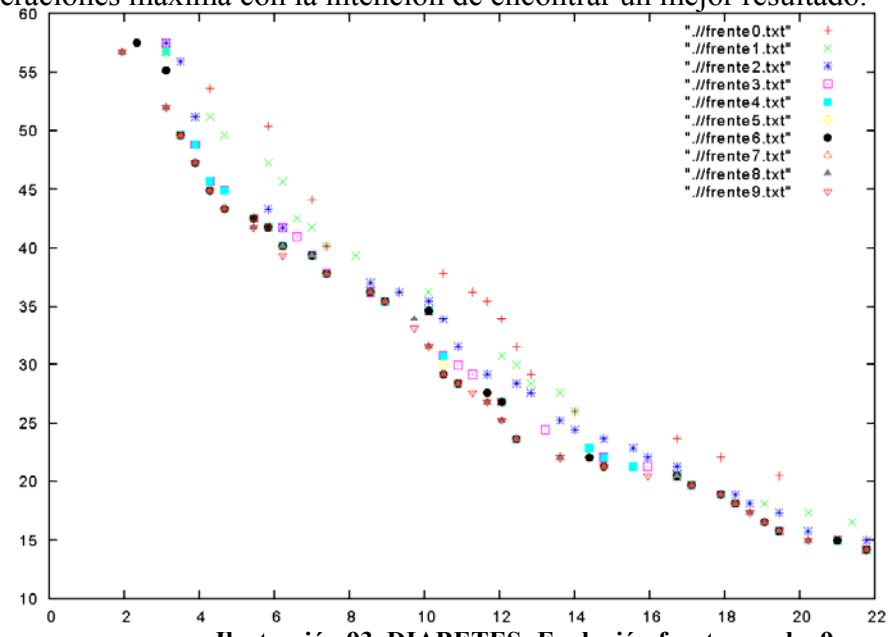


Ilustración 93. DIABETES: Evolución frente prueba 9

Esta prueba a pesar de no observarse en la figura 94, no obtienen los resultados que se esperaban, es decir, no superan los porcentajes ni en test ni en validación de las pruebas realizadas sobre un objetivo.

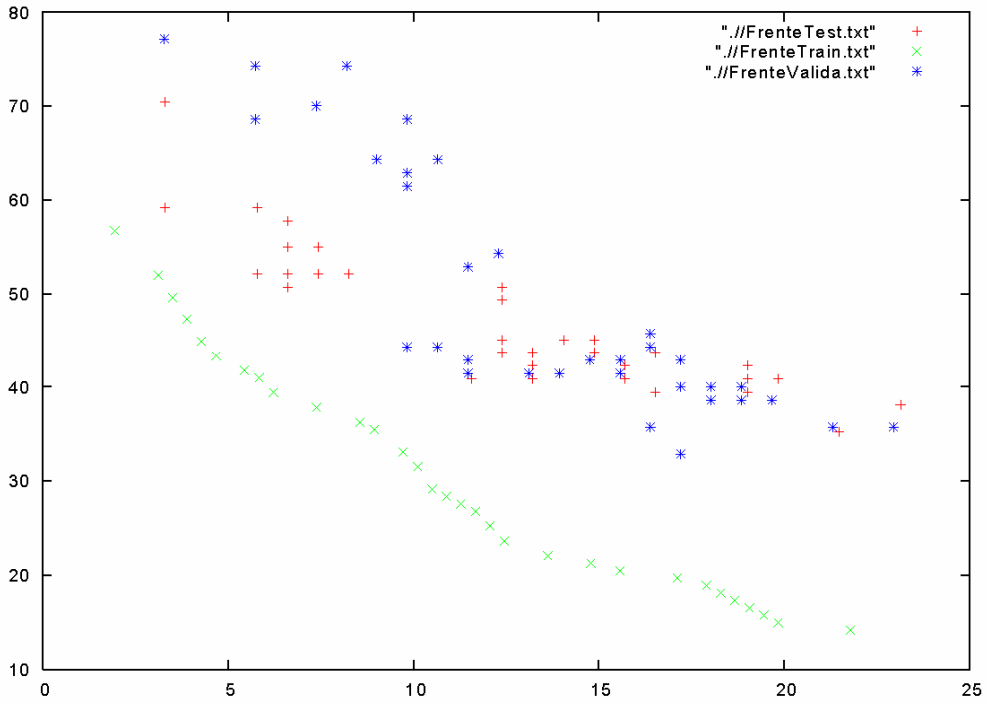


Ilustración 94. DIABETES: Frente final prueba 9

6.6.2.10. Prueba 10

Para la décima prueba se utilizó una población de 24 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el segundo caso, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases, y además se ha utilizado como fitness el conjunto de test. Para esta prueba, se ha cambiado el conjunto de fitness, con el fin de obtener mejores resultados, además se ha decidido utilizar una población pequeña con el mismo fin.

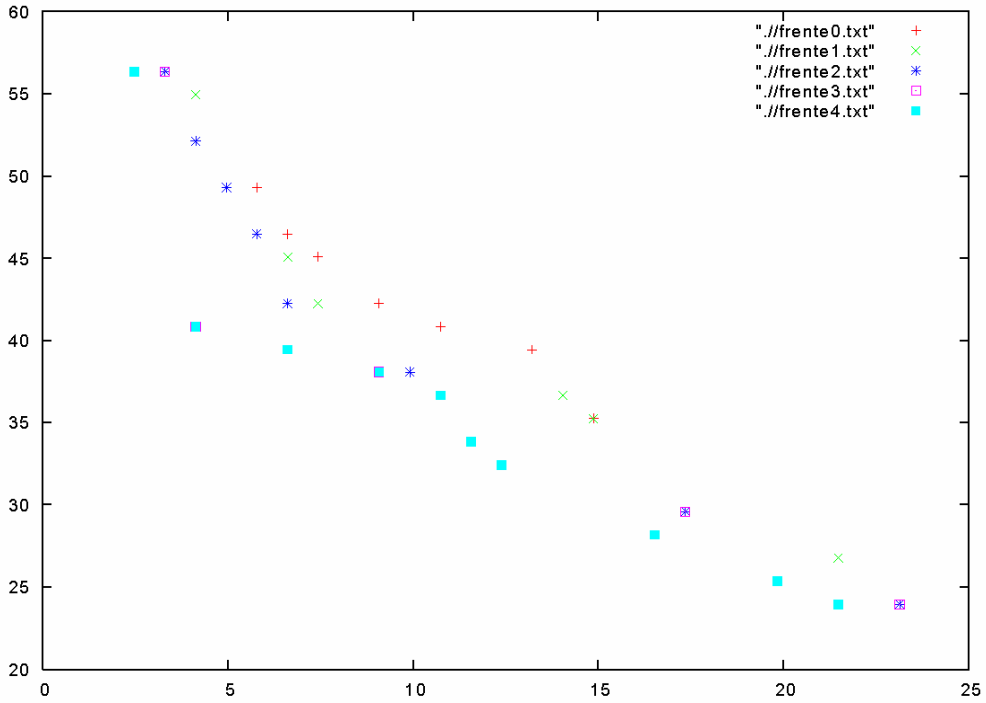


Ilustración 95. DIABETES: Evolución frente prueba 10

Los resultados, como por otro lado cabría esperar ofrecen un mejor resultado en test que cualquier otra prueba anterior. Sin embargo, para el conjunto de validación, si bien los resultados son mejores que cualquier otra prueba sobre el caso 2, no superan a los experimentos realizados con el caso 1. Como curiosidad decir que el frente de entrenamiento y test son el mismo y por tanto en la figura 96 aparecen los mismos puntos pintados.

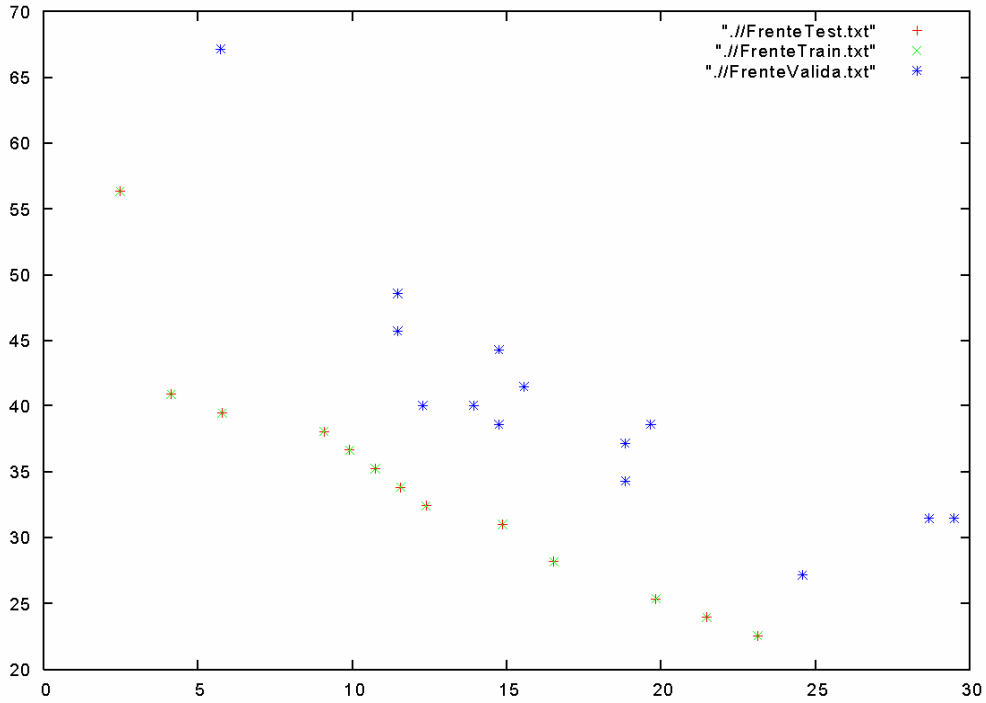


Ilustración 96. DIABETES: Frente final prueba 10

6.6.2.11. Resultados finales

Aunque se han mejorado los resultados iniciales tanto para los conjuntos de entrenamiento como para test, sin embargo para validación no se ha logrado encontrar un mejor resultado comparado con los resultados del perceptron simple.

Si comparamos los resultados de las pruebas unos con otras se puede ver como utilizando el caso 2 se obtiene un mejor resultado en el conjunto de entrenamiento, aunque este resultado no se refleja en test y validación. Además, y como es lógico utilizando el caso 3 se obtiene un mejor resultado en test, ya que utiliza este conjunto para obtener el fitness. Estos resultados sin embargo, no se pueden considerar buenos.

Caso	**Pob.**	**Gen.**	**Ite.**	**Obj0**	**Obj1**	**Tot.**	**%Tot**	**Test**	**%Test**	**Valid**	***%Val***
1	**48**	**100**	**300**	11	70	70	18,22	40	20,83	46	*23,95*
1	**100**	**100**	**300**	7	75	75	19,53	42	21,87	50	*26,04*
1	**48**	**100**	**500**	3	83	83	21,61	43	22,39	46	*23,95*
1	**100**	**100**	**500**	5	76	76	19,79	43	22,39	45	*23,43*
1	**100**	**200**	**500**	6	74	74	19,27	43	22,39	46	*23,95*
2	**48**	**50**	**300**	28	41	69	17,96	44	22,91	49	*25,5*
2	**100**	**100**	**300**	23	44	67	17,44	42	21,87	49	*25,5*
2	**200**	**100**	**300**	24	42	66	17,18	42	21,87	47	*24,47*
2	**100**	**100**	**500**	40	26	66	17,18	43	22,39	51	*26,56*
3	**24**	**50**	**300**	5	29	-	-	34	17.7	46	23.95

Tabla 16. Resultados diabetes

Si tomamos el mejor resultado de los obtenidos en test la primera prueba obtiene unos resultados de 18.22, 20.83 y 23.95 en porcentaje de error para los conjuntos de entrenamiento, test y validación respectivamente. Si tomamos los valores para la diabetes de grado polinomial dos de los resultados de la literatura que también utilizaron functional link [Sierra01] y que son 22.01, 19.37 y 21.92 para los conjuntos de entrenamiento, test y validación respectivamente vemos que los resultados obtenidos son similares en ambos casos.

6.6.3. Cuadrático

Para el dominio cuadrático se parte de un resultado de 66.14, 62.66 y 62.59 con el perceptron simple, el cuál es el que se deberá superar. En este caso sí que sabemos que estos valores serán superados, debido a la naturaleza propia de los datos, ya que han sido generados artificialmente, y la superficie que separa las clases es una expresión lineal de atributos producto, que es el mismo tipo de expresión que el sistema es capaz de aprender. Por tanto, el problema consiste en que el sistema redescubra esa superficie de separación. Por lo que, al aplicar functional link sobre ellos, el resultado debería ser bastante mejor que simplemente utilizando los atributos originales.

6.6.3.1.Prueba 1

Para la primera prueba se utilizó una población de 48 individuos y un número de generaciones del NSGAII de 50, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer objetivo, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En esta primera prueba podemos observar como el frente evoluciona muy rápidamente, llegando incluso a una solución en el último frente en la que el porcentaje de fallos es cero. Se puede ver también que el frente se encuentra muy distribuido y con muchos puntos.

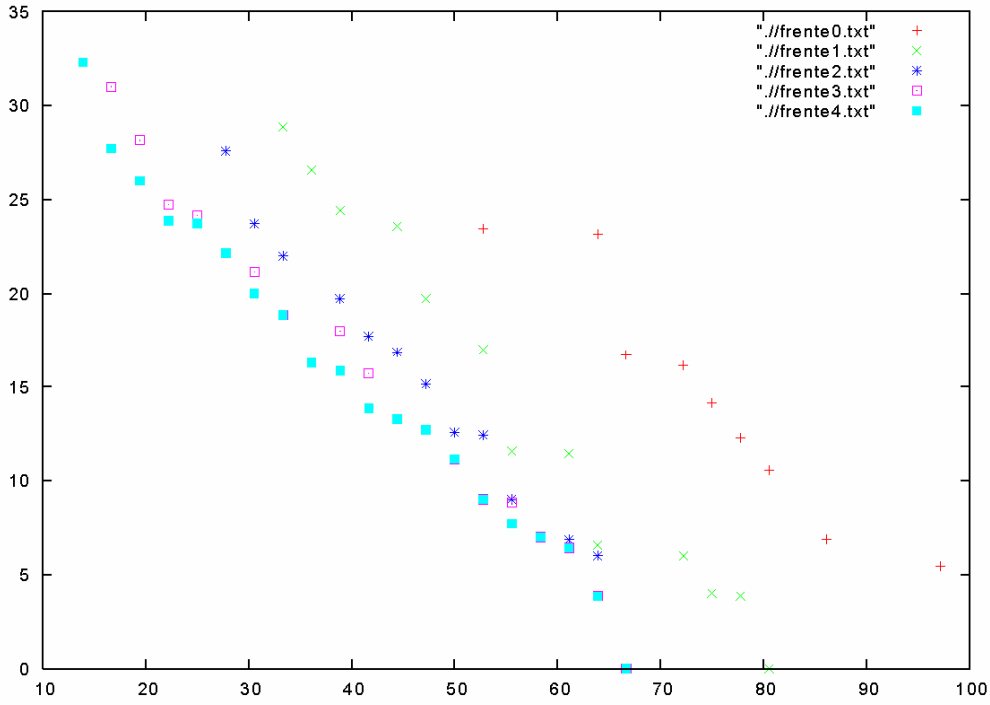


Ilustración 97. CUADRÁTICO: Evolución frente prueba 1

En el frente que se aprecia en la figura 98 se puede apreciar como existe una ligera separación entre los frentes de test y validación con el de entrenamiento, y siempre por encima de este último. También se puede claramente como el mejor punto en test también es el mismo que para entrenamiento y es el que se encuentra más a la

derecha de la gráfica y también más abajo en la misma, sobre el 68% de atributos escogidos.

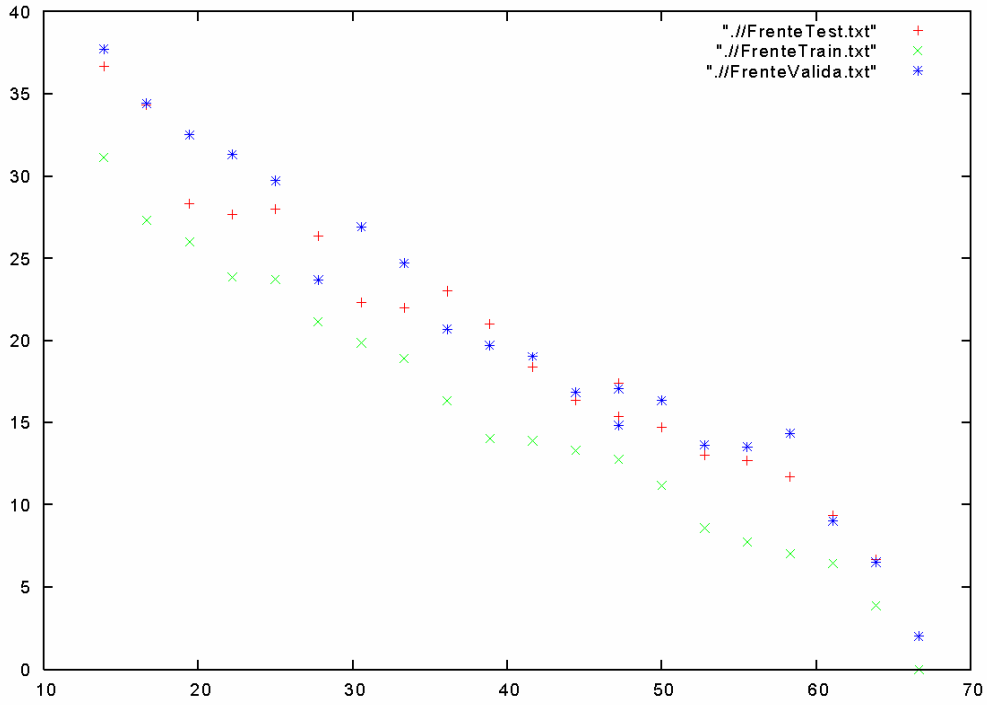


Ilustración 98. CUADRÁTICO: Frente final prueba 1

6.6.3.2.Prueba 2

Para la primera prueba se utilizó una población de 148 individuos y un número de generaciones del NSGAII de 100, utilizando como iteraciones máxima 300. También cabe resaltar que la prueba se realizó sobre el primer objetivo, es decir, el objetivo de minimizar el número de atributos usado, así como el porcentaje de fallos total. En esta prueba se ha decidido aumentar tanto el tamaño de la población como el número de generaciones del NSGAII para intentar llegar a una mejor solución. Se puede observar en la figura 99 como el frente evoluciona rápidamente al principio y como al final también se encuentran mejores soluciones, aunque de una manera más lenta.

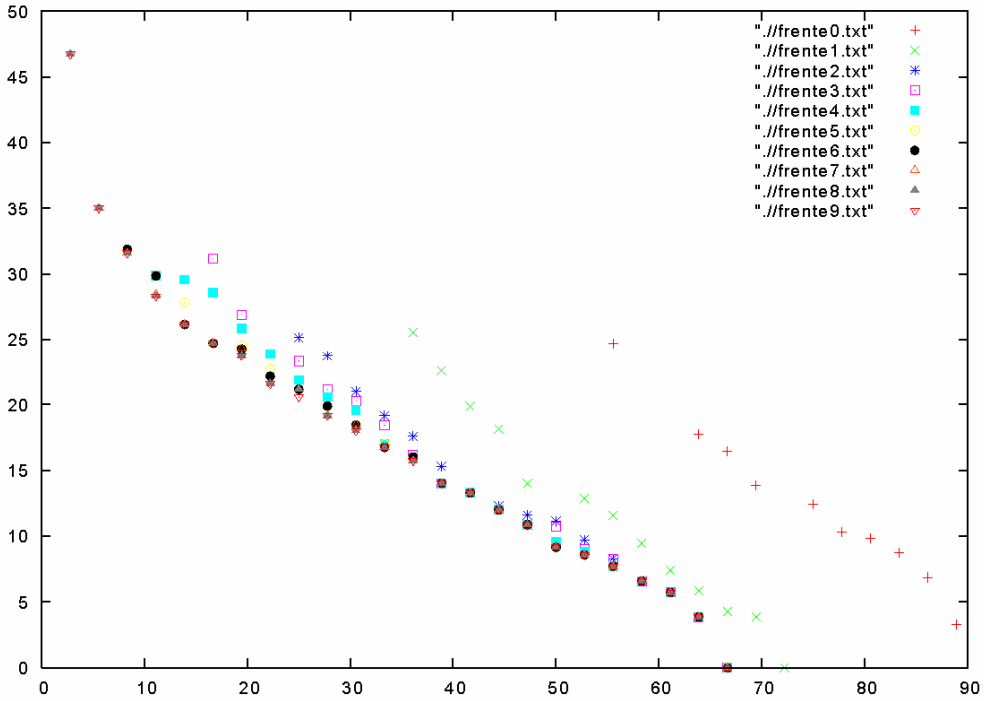


Ilustración 99. CUADRÁTICO: Evolución frente prueba 2

Como se puede apreciar en la figura 100 existen más puntos por la zona izquierda que en la prueba anterior, aunque el mejor punto coincide con el de la prueba anterior.

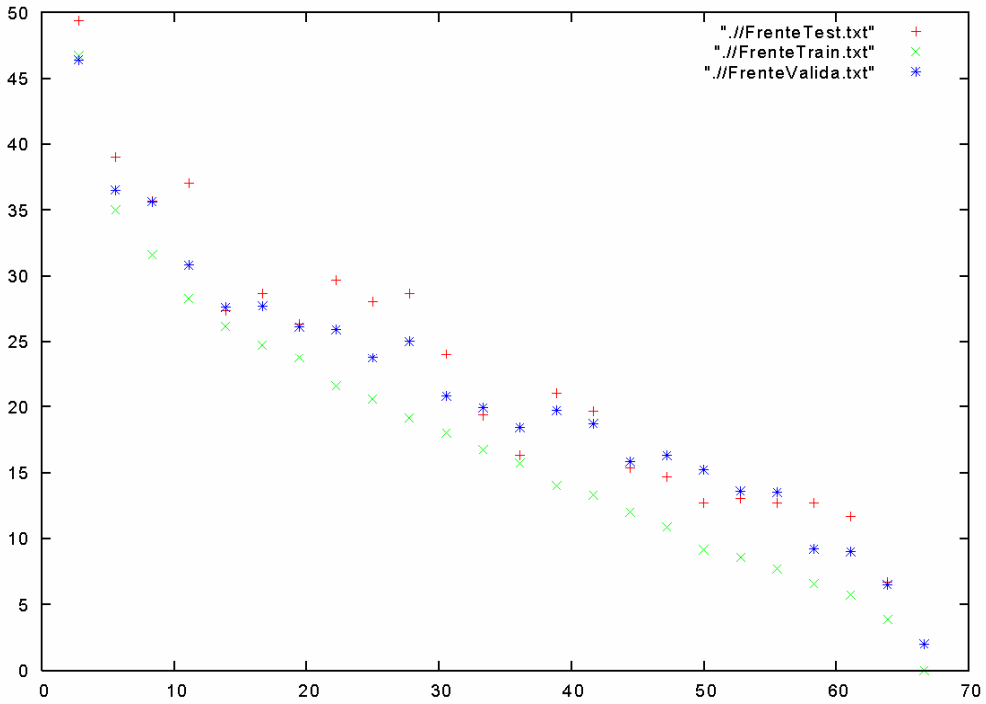


Ilustración 100. CUADRÁTICO: Frente final prueba 1

6.6.3.3.Resultados finales

Sobre este dominio la mejora obtenida ha sido muy grande, ya que hemos pasado del 63% a aproximadamente al 98% de tasa de aciertos. Como se puede ver en ambas pruebas se eligió el mismo punto como mejor resultado, y también se puede ver que se realizaron pocas pruebas debido a los buenos resultados que se obtuvieron. El tiempo si bien en la primera prueba no supero la media hora, para la segunda prueba se necesito cerca de la hora y media de ejecución del algoritmo.

Caso	**Pob.**	**Gen.**	**Ite.**	**Obj0**	**Obj1**	**Total**	**%Tot**	**Test**	**%Test**	**Valid.**	***%Valid.***
1	**48**	**50**	**300**	24	0	0	0	6	2	20	*2*
1	**148**	**100**	**300**	24	0	0	0	6	2	20	2

Tabla 17. Resultados Cuadrático

Al tener este dominio teórico una separación de los datos no lineal, y como nuestro software realizado trata precisamente de realizar separaciones de los datos para obtener mejores resultados, resulta adecuado utilizar esta técnica para este dominio de manera que con ello se ha comprobado que la idea funciona correctamente y efectivamente en dominios adecuados va a mejorar notablemente los resultados ofrecidos por el clasificador lineal.

7. CONCLUSIONES

En este proyecto se ha desarrollado la idea de evolucionar *Functional Link Networks* (FLN) mediante un algoritmo genético multiobjetivo (NSGAII). FLN son una manera sencilla de incorporar no linealidades a un clasificador base lineal (perceptron simple), de manera que se entrenan rápidamente, por lo que son adecuados en un contexto evolutivo. El enfoque multiobjetivo permite obtener un frente diverso de modelos FLN's en función de los casos propuestos. Se han considerado dos casos que diferían en los objetivos utilizados. En el primer caso se minimiza el error y el número de atributos usados por el modelo, mientras que en el segundo caso se minimiza el error de las distintas clases del problema de clasificación. El método ha sido analizado experimentalmente en varios dominios: Clasificación en brain-computer interfaces (BCI), dos dominios de la base de datos de Proben preparados por Prechelt [Prechelt94] como son el cáncer y la diabetes y un dominio sintético cuadrático.

Cuando se realiza un trabajo de investigación como el realizado en el presente proyecto, nadie en principio puede conocer cuales serán los resultados de la idea que se va a realizar o cuales serán los problemas con los que pueda enfrentarse. Realizar este proyecto no tiene nada que ver por ejemplo con la realización de un sistema que trate de que tenga que ver con el diseño de bases de datos o páginas web. Utilizar por primera vez una técnica puede requerir que al final dicha idea no obtenga los resultados que se esperaban o por el contrario se convierta en un éxito.

Al comienzo del presente proyecto no pensaba cual podrían ser los resultados finales de esta idea, ya que aunque los resultados hubiesen sido malos sobre los diferentes dominios, lo importante era poder comprobar como una idea, la de evolucionar FLN con un enfoque multiobjetivo, podía ser usada para resolver problemas de clasificación y para ello, era necesario un dominio y por ello se decidió utilizar uno que se encuentra actualmente en plena investigación como es el de BCI.

Centrándonos en las pruebas realizadas sobre el dominio de BCI observamos cómo se realizaron numerosos cambios en la idea inicial, que con ellos se fueron observando mejoras en los resultados finales. El hecho de realizar una primera selección sobre los atributos originales y luego realizar otra selección sobre los atributos producto se debió principalmente a la incapacidad computacional para poder soportar una cantidad tan grande de atributos.

Por último, se cambió el clasificador base del algoritmo multiobjetivo a tres perceptrones simples con la arquitectura One-against-all y este cambio consiguió mejorar los resultados que se obtenían usando sólo dos perceptrones sobre el dominio de BCI.

Si pensamos un poco en los algoritmos utilizados en la experimentación realizada con Weka en el Anexo "Estudio datos BCI" vemos como el mejor de todos los algoritmos es un separador lineal (Logistic Regresión). Con ello, y vistos los resultados de este proyecto, podemos llegar a la conclusión de que con los datos de BCI que se han manejado, los mejores clasificadores son los lineales y no merece la pena añadir no linealidades. Es cierto que en la competición se consiguen mejores resultados que con los clasificadores lineales, pero los medios usados para la mejora añaden conocimiento del dominio. Por ejemplo, el ganador de la competición incorpora un mecanismo que

detecta transiciones entre pensamientos. Esta idea (y otras usadas en la competición) van algo mas allá del mero uso de clasificadores (lineales o no). Por tanto, la idea de dar un enfoque multiobjetivo a la evolución de FLN no va a mejorar los resultados en caso de que el problema tenga características lineales y el clasificador base escogido sea preciso.

Sin embargo, como se puede ver en las pruebas realizadas con dos perceptrones simples combinados sobre BCI (caso de clasificador base poco preciso) o mucho mejor sobre el dominio sintético cuadrático (caso de dominio no lineal), la idea desarrollada va a mejorar en mayor o menor medida los resultados obtenidos por el clasificador lineal base.

Aparte del dominio BCI, la idea también ha sido probada en dos dominios como el cáncer y la diabetes que nos han permitido comparar nuestra aproximación multiobjetivo con una aproximación evolutiva monoobjetivo ya existente en la literatura [Sierra01]. Se ha comprobado que los resultados son equivalentes en términos de precisión (porcentaje de aciertos), pero el método monoobjetivo requería de mecanismos y ajustes de un parámetro para forzar la evolución de modelos simples que no sobreadaptaran, mientras que este método no requiere de dichos ajustes, puesto que obtiene un frente de modelos FLN's, del que selecciona el mejor mediante un conjunto de test.

Por último, también se ha probado el método en un dominio sintético, cuya frontera de separación es cuadrática. Este método multiobjetivo ha conseguido reencontrar dicha frontera de separación con un error muy cercano al 0% de manera que se ha comprobado que esta idea funciona correctamente cuando las condiciones son adecuadas.

8. LÍNEAS FUTURAS

El campo del aprendizaje automático multiobjetivo es un tema de reciente interés y por tanto existen muchas líneas de investigación abiertas. En el marco de este proyecto, existen aspectos sobre los cuales se podría avanzar o investigar. A continuación se detallan algunos de los aspectos más inmediatos o directos a abordar.

En primer lugar (refiriéndonos en concreto al algoritmo genético multiobjetivo utilizado y debido a las restricciones computacionales existentes actualmente y al desarrollo actual de los multiprocesadores), se podría pensar en utilizar esta capacidad para mejorar el algoritmo genético multiobjetivo de manera que se convirtiera en un algoritmo genético multiobjetivo paralelo. De esta forma se utilizan diversos conjuntos de poblaciones que se evalúan en distintos procesadores; posteriormente y en función del algoritmo se podrían aplicar cualquiera de los diferentes modelos poblacionales existentes [Gorges90].

- El modelo de islas, con poblaciones separadas y migración, siendo cada isla un AG con población global.

- El modelo stepping stone, que cuenta con islas organizadas de acuerdo a una estructura de vecindades no todas contiguas, lo cual limita el intercambio de material genético entre subpoblaciones.

- El modelo de vecindades, en el cual se define una relación de adyacencia sobre los individuos, los cuales pueden interactuar exclusivamente con sus vecinos.

Existen actualmente equipos de trabajo que están investigando sobre estos temas [Fernández02] y de hecho se pueden encontrar en la literatura algunos algoritmos genéticos multiobjetivo ya realizados, incluso sobre el algoritmo NSGAII [Nesmachnow03]. De manera que la ampliación podría consistir en utilizar uno de estos algoritmos genéticos paralelos multiobjetivo para convertir el proyecto en la evolución de functional link networks con un enfoque multiobjetivo paralelo.

En lo referente al software desarrollado, un posible trabajo futuro sería preparar dicho software para que pudiera utilizarse en dominios de clasificación con más de tres clases, ya que este software ha sido preparado para dominios con dos o tres clases. El único cambio que se debería realizar sería sobre los perceptrones simples, ya que sería necesario utilizar otra combinación de perceptrones de manera que pudieran hacer frente a una separación de clases mayor que tres.

En esta misma línea de trabajo y en lo que respecta al software desarrollado se podrían incorporar diferentes implementaciones del perceptron simple para abordar problemas de clasificación de manera que sea la persona la que elija la implementación que desea utilizar. O también se podría plantear el software de manera que realizara pruebas sobre diferentes clasificadores con los atributos originales y después fuera el mismo programa el que escogiera la implementación que mejor resultado obtiene en base a esas pruebas y fuera el utilizado para realizar la ejecución del algoritmo genético multiobjetivo. Se podrían plantear también en las diferentes implementaciones de los

clasificadores posibles mejoras que consistirían en variar las características propias del clasificador para integrarlo en el software, y que podrían consistir, por ejemplo, en utilizar la combinación de dos perceptrones simples utilizando la regla de aprendizaje de Widrow-Hoff o utilizando la razón de aprendizaje α.

En lo referente al clasificador utilizado en este proyecto (Functional Link Networks), se podría pensar en aumentar el grado polinomial usado o incluso incorporar otros clasificadores no lineales como el perceptron multicapa. La eficiencia de aumentar el grado polinomial va a depender de ciertos factores, como:

- El dominio sobre el que se ejecute. En función del número de atributos originales, el aumentar el grado polinomial puede implicar que la longitud del cromosoma sea demasiado elevada.

- La capacidad de cómputo del hardware. Dependerá del tiempo que se tarde en realizar la evaluación de un individuo.

- La capacidad de paralelización del algoritmo genético multiobjetivo. Como ya se ha comentado anteriormente se podrían realizar varias evaluaciones de individuos a la vez, y de esta manera en lugar de bajar el tiempo de ejecución se podría aumentar la complejidad del problema utilizando alguna de las mejores que se han descrito sobre FLN.

También se podría comprobar la mejora obtenida al realizar un enfoque multiobjetivo, de manera que se desarrollará el mismo algoritmo pero con un enfoque monoobjetivo, para comprobar que ha aportado el enfoque multiobjetivo en los diferentes dominios a los resultados obtenidos.

Por supuesto se podrían realizar más experimentos sobre otros dominios de experimentación, de manera que se deberían utilizar dominios donde las separaciones de los datos fueran no lineales en donde el éxito de nuestra técnica está prácticamente asegurado, o en los dominios donde la clasificación lineal no sea adecuada para comprobar cómo se comporta este sistema en esos dominios.

Si nos centramos ahora en el dominio de la BCI, una primera ampliación de este proyecto podría ser la realización de pruebas con otros datos de la misma naturaleza, bien con datos recogidos en situaciones similares a los utilizados en este proyecto, bien con datos procedentes de otras competiciones.

9. BIBLIOGRAFÍA

[Alfaro05] Clasificación de señales de electroencefalograma usando programación genética. Eva Alfaro-Cid, Anna I. Esparcia-Alcázar, Ken Sharman. Instituto Tecnológico de Informática. Ciudad Politécnica de la Innovación. Universidad Politécnica de Valencia.

[Coello99] COELLO COELLO, Carlos A. ((A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques.)) Knowledge and Information Systems. An International Journal, 1, no 3, (1999).

[Coello01] Carlos A. Coello Coello and Gregorio Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, pages 126–140, London, UK, 2001. Springer-Verlag.

[Darwin1859] DARWIN, Charles Robert. The Origin of Species by Means of Natural Selection, or The Preservation of Favoured Races in the Struggle for Life. Penguin Books Ltd., Nueva York, EE. UU., 1959. Publicado originalmente en 1859.

[Deb94] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation, 1994.

[Deb99] ((Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design.)) En Evolutionary Algorithms in Engineering and Computer Science (Kaisa Miettinen, Marko M. Makela, Pekka Neittaanmaki, y Jacques Periaux, eds.), John Wiley & Sons, Ltd, Chichester, Reino Unido, 1999.

[Deb00] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[Eri01] Mark Erickson, Alex Mayer, and Jeffrey Horn. The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, First International Conference on Evolutionary Multi-Criterion Optimization, pages 681–695. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[Fernández02] Equipo de Algoritmos Evolutivos Multiobjetivo Paralelos (Team Algorithm of pMOEAs) José Manuel Fernández Giangreco Universidad Católica de Asunción. Paraguay.

[Fogel99] Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming. John Wiley & Sons, Inc., Nueva York, 1999.

[Fonseca93] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. 1993.

[Gho92] J. Ghosh and Y.Shin, "Efficient higher order neural networks for classification and function approximation" Int. J. Neural Syst, 1992.

[Gorges90] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. Proceedings of 1st PPSN'90, pp. 150-159, 1990.

[Holland75] HOLLAND, John H. Adaptation in Natural and Artificial Systems. Ann Arbor: University Press, 1975.

[Hor91] K. Hornik, "Approximation capabilities of multilayer feddforward Networks" Neural Networks 1991.

[Horn94] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pages 82–87, Piscataway, New Jersey, 1994. IEEE Service Center.

[Knowles00] Joshua D. Knowles and David Corne. Approximating the nondominated front using the pareto archived evolution strategy. Evolutionary Computation, 8(2):149–172, 2000.

[Mendel1901] MENDEL, Gregor Johann. ((Experiments in plant hybridisation.)) Journal of the Royal Horticultural Society, no 26, (1901), 1.32.

[Michie94] D. Michie, D.J. Spiegelhalter, C.C. Taylor (eds). Machine Learning, Neural and Statistical Classification, 1994

[Mill04] Millán, J. del R. On the need for on-line learning in brain-computer interfaces Proc. Int. Joint Conf. on Neural Networks., 2004.

[Nesmachnow03] Una Versión Paralela del Algoritmo Evolutivo para Optimización Multiobjetivo NSGA-II y su Aplicación al Diseño de Redes de Comunicaciones Confiables. Sergio Nesmachnow. Centro de Cálculo, Instituto de Computación. Facultad de Ingeniería. Universidad de la República, Uruguay.

[Osyczka85] A. Osyczka. Multicriteria optimization for engineering design, chapter Design optimization. Academic Press, Cambridge, 1985.

[Pao89] Y.H. Pao. Adaptive Pattern Recognition and Neural Networks. Addison-Wesley. 1989.

[Pao94] Y.H. Pao, G. H. Park and D. J. Sobajic, "Learning and generalization characteristics of the randomvector functional link net" Neurocomputing 1994.

[Prechelt94] L. Prechelt, "Proben1. A set of neural network benchmark problems and benchmarking rules," Tech. Rep. 21/94, Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Sept. 1994.

[Romero05] Evolutionary Design of a Brain-Computer Interface. G. Romero, M.G. Arenas, P.A. Castillo, and J.J. Merelo. Department of Architecture and Computer

Technology, University of Granada, ETSII. Department of Computer Science, University of Jaén, EPS.

[Rosenberg67] ROSENBERG, R. S. Simulation of genetic populations with biochemical properties. Tesis Doctoral, University of Michigan, Ann Harbor, Michigan, EE. UU., 1967.

[Schaffer84] SCHAFFER, J. David. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. Tesis Doctoral, Vanderbilt University, 1984.

[Sierra01] "Evolution of functional link networks." A. Sierra, J. A. Macías, and F. Corbacho. IEEE Transactions on Evolutionary Computation, vol. 5.

[Sob88] Sobajic, D. J .Artificial Neural Networks for Transient Stability Assessment of Electric Power Systems.. UMI Dissertation Information Service. 1988

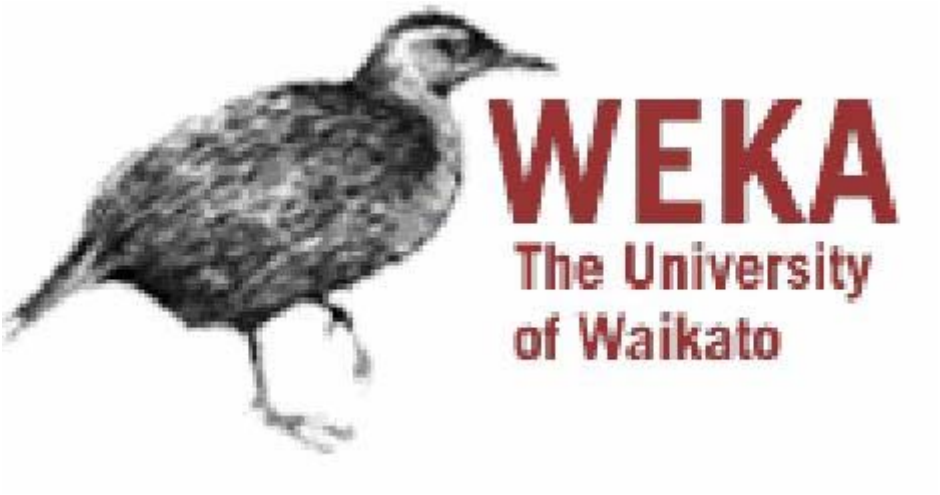
[Uge97] Ugena. A.M .Arquitectura de Redes Neuronales con Ligadura Funcional. Ph D. Thesis. Departamento de Matemática Aplicada. Universidad Politécnica de Madrid. 1997

[Weismann1893] WEISMANN, August Friedrich Leopold (ed.) The Germ Plasm: A Theory of Heredity. Scott, 1893.

[Zitzler99] Zitzler, Eckart y Thiele, Lothar; "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach", IEEE Transactions on Evolutionary Computation, Vol. 3, Nº 4, 1999.

[Zitzler01] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer engineering and networks laboratory, Swiss Federal Institute of Technology, Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, may 2001.

ANEXO I. ESTUDIO DATOS BCI



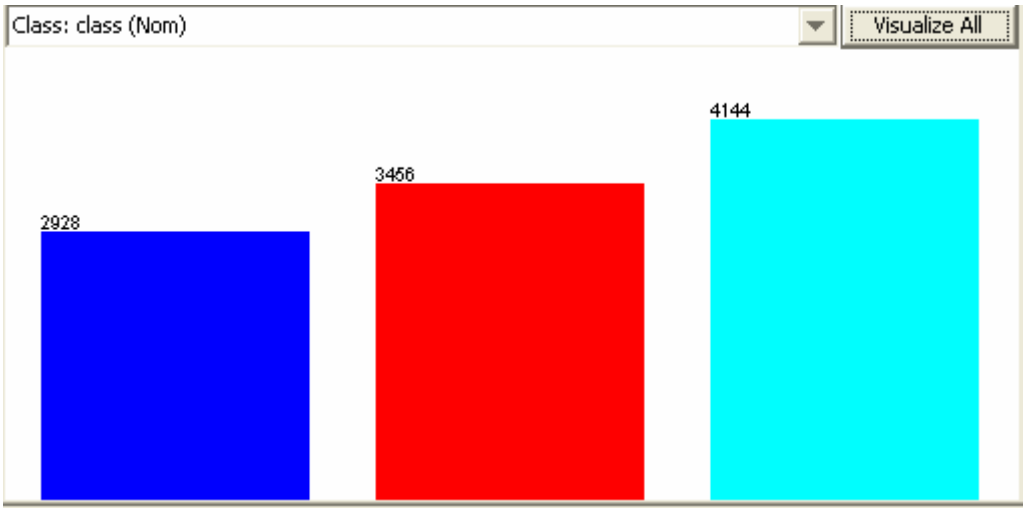
HERRAMIENTAS DE LA INTELIGENCIA ARTIFICIAL

Ángel Carrasco Fernández **NIA: 100047485**

1. Exploración de los datos

1.1. Sujeto 1

- **¿Estamos trabajando con muestras con clases desequilibradas?**



El gráfico representa el número de instancias que han sido clasificadas para cada una de las clases, como se puede observar de los 10524 que existen en total, existen 4144 con la clase 7, y con 3456 datos de la clase 3, y 2928 de la clase 2. Con este gráfico se puede ver que aunque la diferencia de las clases no sea abismal, si existe un desequilibrio notable, sobre todo entre la clase 7 y 2.

- **¿Qué algoritmos parecen trabajar mejor con estos datos? ¿Se obtienen buenos resultados?**

A continuación se muestra una tabla en la que aparecen las pruebas realizadas con los distintos algoritmos y los resultados que se han obtenido con cada uno de ellos, de forma que ordenados por el porcentaje de aciertos obtenidos. Si no se pone ninguna anotación en el algoritmo utilizado significa que se ha utilizado el algoritmo descrito con los parámetros que aparecen por defecto en Weka.

ALGORITMO UTILIZADO	PORCENTAJE ACIERTO
LOGISTIC	74.3296 %
BAGGING	73.352 %
BAYESNET	71.8156 %
RBFNETWORK	71.5642 %
NAIVEBAYES	71.2011 %
NAIVEBAYESSIMPLE	71.1732 %
J48	63.0447 %
ZEROR	40.6704 %

- **¿Qué indica la selección de atributos? ¿Hay muchos atributos significativos o pocos?**

Para realizar la selección de atributos se han utilizado alguna de las técnicas existentes, siendo los resultados diversos en función de la técnica utilizada, de manera que en una prueba sólo se seleccionaban 11 atributos, mientras que en otra se seleccionaban 47.

1ª Prueba

=== Run information ===

Evaluator: weka.attributeSelection.InfoGainAttributeEval
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation: eeg-train-test1-subject1-psd1_2
Instances: 10528
Attributes: 97
=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 97 class):
Information Gain Ranking Filter

Ranked attributes:
0.31172 38 CP1-2
0.24702 27 C4-3

0.24039 26 C4-2
0.22876 2 C3-2
0.22083 3 C3-3
0.17997 39 CP1-3
0.11932 8 C3-8
0.1166 14 Cz-2
0.11225 25 C4-1
0.07804 31 C4-7
0.06943 7 C3-7
0.05304 37 CP1-1
0.05092 9 C3-9
0.04406 32 C4-8
0.04007 12 C3-12
0.03991 63 P3-3
0.03991 62 P3-2
0.0399 79 Pz-7
0.03791 84 Pz-12
0.03735 15 Cz-3
0.03664 44 CP1-8
0.03517 70 P3-10
0.0345 68 P3-8
0.03411 89 P4-5
0.03369 88 P4-4
0.03355 45 CP1-9
0.03267 91 P4-7
0.03225 80 Pz-8
0.03123 33 C4-9
0.03114 21 Cz-9
0.03092 40 CP1-4
0.0307 57 CP2-9
0.03034 64 P3-4
0.03003 69 P3-9
0.02986 94 P4-10
0.02923 17 Cz-5
0.02911 28 C4-4
0.0289 58 CP2-10
0.0286 43 CP1-7
0.02772 61 P3-1
0.02749 1 C3-1
0.02726 16 Cz-4
0.02719 81 Pz-9
0.0269 83 Pz-11
0.02655 46 CP1-10
0.02614 50 CP2-2
0.02579 90 P4-6
0.02569 53 CP2-5
0.02527 65 P3-5
0.02515 6 C3-6
0.02515 96 P4-12
0.02514 72 P3-12

0.02441 66 P3-6
0.02394 19 Cz-7
0.02388 85 P4-1
0.02342 71 P3-11
0.02313 82 Pz-10
0.02311 22 Cz-10
0.02309 23 Cz-11
0.02298 52 CP2-4
0.02274 24 Cz-12
0.02236 47 CP1-11
0.02196 20 Cz-8
0.02182 77 Pz-5
0.02165 59 CP2-11
0.02129 95 P4-11
0.02105 4 C3-4
0.02068 54 CP2-6
0.02067 41 CP1-5
0.02043 34 C4-10
0.02024 87 P4-3
0.02008 78 Pz-6
0.0198 42 CP1-6
0.01948 92 P4-8
0.01866 73 Pz-1
0.01856 56 CP2-8
0.01847 60 CP2-12
0.01796 76 Pz-4
0.01795 93 P4-9
0.01792 29 C4-5
0.01745 11 C3-11
0.01707 36 C4-12
0.01647 5 C3-5
0.01545 67 P3-7
0.01371 48 CP1-12
0.01347 13 Cz-1
0.01319 75 Pz-3
0.01295 55 CP2-7
0.01222 35 C4-11
0.01111 86 P4-2
0.01013 18 Cz-6
0.00852 10 C3-10
0.0063 30 C4-6
0.00606 74 Pz-2
0.00234 49 CP2-1
0 51 CP2-3

Selected attributes: 38, 27, 26, 2, 3, 39, 8, 14, 25, 31, 7, 37, 9, 32, 12, 63, 62, 79, 84, 15,44,70,68,89,88,45,91,80,33,21,40,57,64,69,94,17,28,58,43,61,1,16,81,83,46,50,90,53,65,6,96,72,66,19,85,71,82,22,23,52,24,47,20,77,59,95,4,54,41,34,87,78,42,92,73,56,60,76,93,29,11,36,5,67,48,13,75,55,35,86,18,10,30,74,49,51 : 96

2ª Prueba

=== Run information ===

Evaluator: weka.attributeSelection.CfsSubsetEval
Search: weka.attributeSelection.GeneticSearch -Z 20 -G 20 -C 0.6 -M 0.033 -R 20 -S 1
Relation: eeg-train-test1-subject1-psd1_2
Instances: 10528
Attributes: 97
=== Attribute Selection on all input data ===

Search Method:
Genetic search.
Start set: no attributes
Population size: 20
Number of generations: 20
Probability of crossover: 0.6
Probability of mutation: 0.033
Report frequency: 20
Random number seed: 1

Initial population
merit scaled subset
0.13899 0.15216 1 2 4 5 8 10 12 16 17 19 20 21 23 24 27 32 35 38 39 46 48 50 52 54 55 57 61 62 63 66 68 69 70 71 73 75 77 78 82 85 89 91 93
0.07257 0.06787 1 4 7 8 9 13 14 19 21 26 32 33 43 44 45 46 48 49 50 51 52 53 54 55 56 60 62 63 70 71 73 74 78 80 83 85 87 88 89 90 91 92 96
0.12693 0.13686 1 3 4 5 6 7 10 12 13 15 16 17 19 20 21 23 26 28 29 34 35 37 38 40 43 46 47 48 49 50 54 55 56 57 61 62 63 66 68 71 72 74 75 76 78 79 81 82 83 85 87 89 93 94 96
0.0826 0.08059 1 4 8 10 11 12 13 14 15 21 22 23 25 26 28 29 30 31 33 34 38 41 46 49 51 53 54 55 57 58 60 62 65 67 69 72 73 76 77 78 80 81 83 84 85 86 87 88 89 90 93 95 96
0.11153 0.11732 5 8 9 10 11 12 15 16 18 19 21 27 35 37 39 41 46 52 57 58 59 60 61 65 66 68 70 73 78 80 81 82 83 85 87 88 89 90 91 92 93 96
0.06626 0.05987 3 5 8 9 11 29 32 39 44 50 51 56 59 61 62 71 74 80 81 84 86 93
0.10278 0.10621 8 14 20 26 28 30 37 43 45 63 66 68 72 73 75 92
0.04205 0.02914 5 18 32 46 58 60 82 83 86 87
0.05899 0.05064 1 5 8 9 11 12 15 23 25 30 31 35 42 44 46 47 48 51 53 56 62 63 69 70 72 76 79 81 93
0.08098 0.07855 1 4 5 7 8 9 10 11 12 13 14 17 18 19 22 23 25 26 29 30 35 37 39 40 41 42 43 47 48 51 54 56 58 59 60 63 64 65 66 67 69 70 71 73 75 77 78 79 80 81 82 83 85 86 87 89 90 91 92 93 95
0.0191 0 40
0.08107 0.07866 1 2 3 4 5 10 11 12 15 16 19 20 22 25 28 30 32 35 38 45 51 56 57 58 61 63 64 65 67 68 70 73 75 76 81 85 87 92 93
0.07693 0.07341 1 4 6 8 25 30 37 53 60 66 67 68 71 74 75 79 93

0.11726 0.12459 1 4 5 6 7 12 17 22 25 26 29 31 38 42 43 45 46 48 49 57 63 64 67 76 78 80 87 88 90
0.07173 0.0668 3 49 52
0.07403 0.06972 1 2 4 7 16 23 26 27 35 46 51 52 64 67 70 74 77 78 84 85 86 88 89 92 95
0.15868 0.17716 1 2 3 4 8 10 11 12 13 16 17 18 21 22 24 26 27 31 32 37 38 39 42 44 45 46 47 53 55 57 58 61 62 67 69 70 74 76 77 78 81 84 87 89 90 94 96
0.11841 0.12605 2 5 9 10 11 15 16 17 18 21 22 23 24 26 27 29 30 31 34 40 41 44 45 46 47 53 55 57 58 59 62 63 64 65 66 67 68 70 72 74 75 76 77 79 82 83 84 85 86 87 89 90 91 92 94 96
0.07083 0.06566 4 6 8 15 16 17 20 22 28 29 30 31 36 38 41 43 44 45 50 51 54 55 56 57 62 63 66 68 70 71 73 74 76 78 80 81 83 84 85 87 88 90 93 94
0.12892 0.13939 1 4 5 8 10 11 12 13 17 20 24 25 26 29 31 35 38 39 41 43 44 45 48 50 55 58 60 61 63 64 68 71 76 81 82 85 86 92 93

Generation: 20
merit scaled subset
0.175 0.29106 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 25 26 27 31 32 37 38 39 42 45 49 53 55 57 61 62 64 66 69 71 72 74 76 78 79 84 87 89 90 94
0.175 0.29106 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 25 26 27 31 32 37 38 39 42 45 49 53 55 57 61 62 64 66 69 71 72 74 76 78 79 84 87 89 90 94
0.1498 0 1 3 4 8 9 11 12 14 16 18 21 24 25 26 27 31 32 37 39 42 45 50 53 55 61 64 66 67 74 76 77 78 79 80 84 87 89 90
0.15189 0.0241 2 4 7 8 11 12 13 14 15 16 18 22 23 24 25 27 32 35 37 38 39 42 44 45 46 49 53 55 57 61 62 64 66 69 71 74 75 76 78 79 84 87 89 90 92 94 96
0.15039 0.00679 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 25 26 31 32 37 39 42 45 46 49 53 55 57 61 62 63 64 66 70 71 72 74 75 76 78 79 83 84 87 89 90 91 94
0.17322 0.27054 1 2 3 4 7 8 9 11 14 15 16 18 20 21 22 24 25 26 27 31 32 37 38 39 42 45 49 53 55 57 61 62 64 66 69 71 72 74 76 78 79 84 87 89 90 92 94
0.17324 0.27076 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 26 27 31 32 37 38 39 42 45 46 48 49 50 53 55 61 63 64 69 76 77 78 79 80 84 87 89 90 94
0.15473 0.05695 2 8 11 12 14 15 16 23 24 25 27 31 32 35 37 38 39 40 42 44 45 46 49 53 55 57 61 62 64 66 69 71 72 74 76 78 79 82 84 85 87 89 90 94
0.15642 0.07646 1 2 3 4 7 8 11 12 14 15 16 18 20 21 22 24 25 26 27 28 31 39 42 44 45 49 50 53 55 57 64 66 74 76 77 78 79 80 84 87 90 92 93
0.15747 0.08854 1 2 3 4 7 8 11 12 14 15 16 18 20 21 22 24 25 26 27 28 31 39 42 44 45 49 50 53 55 57 64 66 74 76 77 78 79 80 84 87 90 92
0.17111 0.2462 1 2 3 4 8 9 11 14 16 18 21 24 25 26 27 31 32 37 38 39 42 49 53 55 56 57 61 62 64 69 72 74 76 78 79 84 87 88 89 90 91 94 95 96
0.17355 0.27436 1 2 3 4 7 8 9 10 11 12 14 15 18 21 22 24 25 26 27 32 37 38 39 42 45 52 53 55 57 61 62 64 66 69 71 74 76 78 79 81 84 87 89 90 96
0.17441 0.28429 1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 18 21 24 25 26 27 31 32 37 38 39 40 42 44 45 49 50 53 55 57 61 63 64 66 69 74 76 77 78 79 84 87 89 90
0.1524 0.03003 2 8 11 12 14 15 16 22 23 24 25 27 30 31 32 35 37 38 39 42 45 49 53 55 56 57 61 62 64 66 69 71 72 73 74 76 78 79 84 87 89 94
0.17397 0.27923 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 25 26 27 31 32 37 38 39 42 45 49 53 55 57 61 62 64 66 69 71 72 73 74 76 77 79 80 84 87 89 90 94
0.15288 0.03556 2 4 8 11 12 14 15 16 18 22 23 24 25 27 31 32 33 35 37 38 39 42 44 45 49 50 53 55 56 61 64 66 71 74 76 78 79 84 87 89 90 93 94

0.17087 0.24334 1 2 3 4 7 8 9 11 12 14 15 16 18 21 22 24 25 26 27 31 32 38 39 42 45 48 49 53 55 57 61 62 64 66 69 71 72 74 76 77 78 79 84 87 89 90 94
0.17111 0.24618 1 2 3 4 8 9 11 12 14 15 16 18 21 22 24 25 26 27 31 32 37 38 39 42 45 49 53 55 57 61 62 64 65 66 68 69 71 72 74 76 78 79 84 87 89 90 94 96
0.16516 0.17735 1 2 3 4 7 8 11 12 14 15 16 17 18 21 22 24 25 27 31 32 36 37 38 39 40 42 44 45 49 50 53 55 57 61 64 65 66 69 74 76 77 78 79 80 84 87 90 92
0.15735 0.08721 2 3 4 7 8 11 12 14 15 16 18 20 21 22 24 25 26 27 28 31 39 42 44 45 49 50 53 55 57 64 66 76 77 78 79 80 83 87 89 90 93

Attribute Subset Evaluator (supervised, Class (nominal): 97 class):
CFS Subset Evaluator
Including locally predictive attributes

Selected attributes: 1, 2, 3, 4, 7, 8, 9, 11, 12, 14, 15, 16, 18, 21, 22, 24, 25, 26, 27, 31,32,37,38,39,42,45,49,50,53,55,57,61,62,64,66,69,71,72,74,76,78,79,84,87,89,90,94: 47

C3-1
C3-2
C3-3
C3-4
C3-7
C3-8
C3-9
C3-11
C3-12
Cz-2
Cz-3
Cz-4
Cz-6
Cz-9
Cz-10
Cz-12
C4-1
C4-2
C4-3
C4-7
C4-8
CP1-1
CP1-2
CP1-3
CP1-6
CP1-9
CP2-1
CP2-2
CP2-5
CP2-7
CP2-9
P3-1
P3-2
P3-4

P3-6
P3-9
P3-11
P3-12
Pz-2
Pz-4
Pz-6
Pz-7
Pz-12
P4-3
P4-5
P4-6
P4-10

3ª Prueba

=== Run information ===

Evaluator: weka.attributeSelection.CfsSubsetEval
Search: weka.attributeSelection.BestFirst -D 1 -N 5
Relation: eeg-train-test1-subject1-psd1_2
Instances: 10528
Attributes: 97
=== Attribute Selection on all input data ===

Search Method:
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 1267
Merit of best subset found: 0.241

Attribute Subset Evaluator (supervised, Class (nominal): 97 class):
CFS Subset Evaluator
Including locally predictive attributes

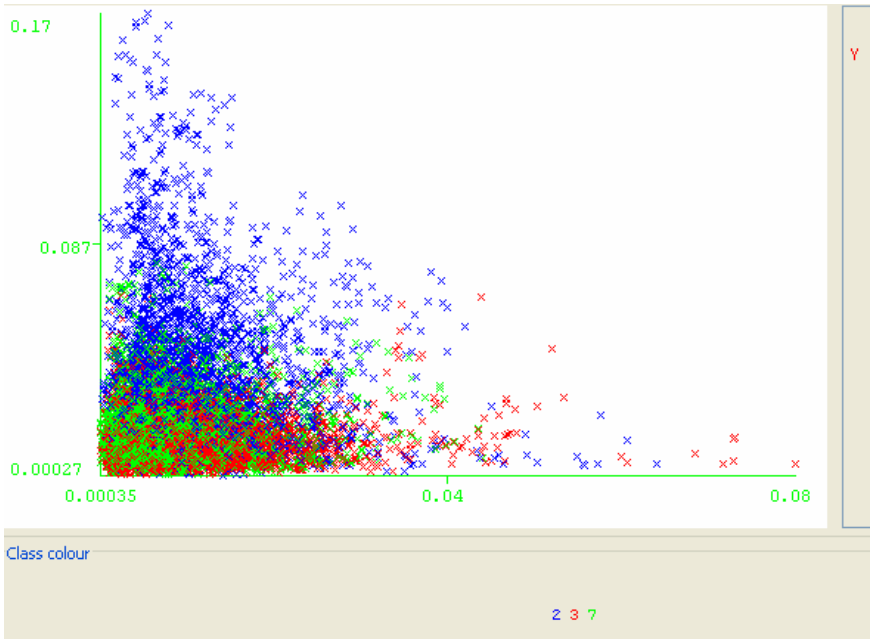
Selected attributes: 2,3,8,14,19,25,26,27,38,39,50 : 11
C3-2
C3-3
C3-8
Cz-2
Cz-7
C4-1
C4-2
C4-3
CP1-2
CP1-3
CP2-2

- **¿Qué ocurre si utilizamos el mejor algoritmo encontrado en conjunción con la selección de atributos? ¿Se mejoran o se empeoran los resultados?**

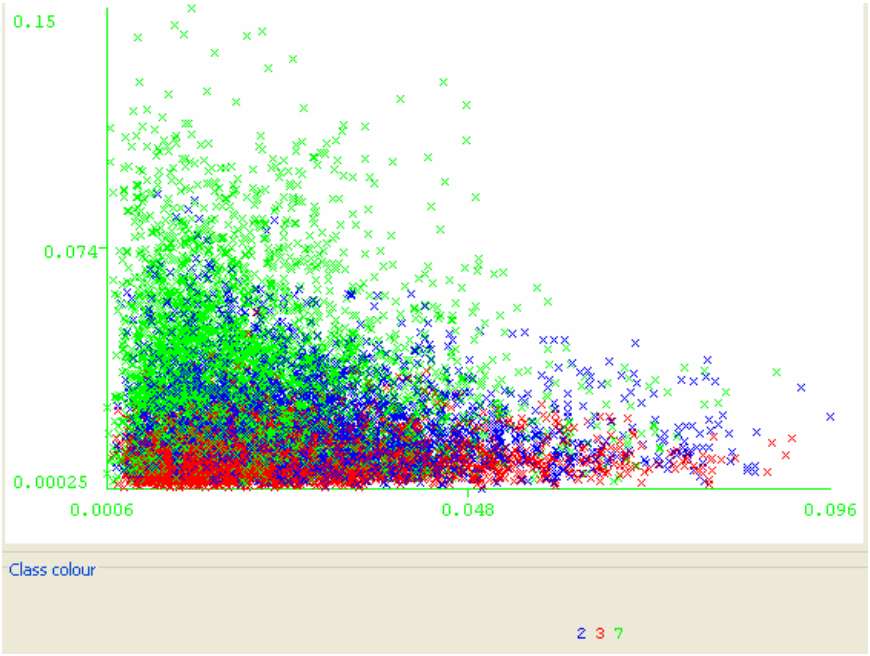
Se ha utilizado la selección con GeneticReserch de manera que se han seleccionado 47 atributos, para después utilizar el algoritmo de Logistic, con el que se ha obtenido un resultado de 75.8659 %, que resulta mejorar el resultado sin selección de atributos. Aunque la mejora no es muy grande, pero se gana en tiempo a la hora de probar el algoritmo.

- **¿Se puede ver de manera gráfica la separación entre las clases utilizando alguna pareja de los atributos seleccionados?**

No se ha logrado encontrar una pareja de atributos que logre separar todas las clases, pero lo que si hemos podido encontrar es un par de correlaciones que logran separar una clase del resto. En este primer gráfico se muestra como los atributos P4-4 y C3-2 son capaces de separar la clase 2 (azul) de las otras dos.



Y en el siguiente gráfico que muestra la correlación entre P3-4 y CP1-2, se puede observar como separan la clase 7 (verde) de una forma más o menos clara.



- **Utilizando el mejor algoritmo, ¿las distintas clases se aprenden de manera desequilibrada? ¿Cuáles se aprenden mejor? ¿Se puede equilibrar el porcentaje de aciertos de las distintas clases de alguna manera?**

A continuación se muestra la matriz de confusión para el algoritmo Logistic con todos los 96 atributos, de manera que se puede ver como se producen fallos en todas las clases, por lo que en este caso no parece adecuado utilizar la matriz de costes para conseguir equilibrar los fallos en las clases.

```
a   b   c   <-- classified as
 733  150  253 |   a = 2
 115  742  131 |   b = 3
  57  213 1186 |   c = 7
```

2. Experimentación

Para realizar la experimentación se ha utilizado el mejor algoritmo con y sin la selección de atributos con GeneticReserch. Por tanto, se realizarán dos algoritmos, más el de ZeroR con los datos de cada uno de los sujetos. De esta experimentación se obtienen los siguientes resultados:

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   3
Resultsets: 7
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       11/06/08 12:47
```

Dataset (1) rules.Ze | (2) funct (3) meta. (4) meta. (5) meta. (6) meta. (7) meta.

---

eeg-train-test1-subject1- (1) 40.67 | 74.33 72.96 73.27 73.88 73.69 75.53
eeg-train-test1-subject2- (1) 40.72 | 62.02 59.22 61.03 62.36 59.98 63.21
eeg-train-test1-subject3- (1) 31.97 | 60.55 99.09 61.15 53.81 98.80 53.53

(v/ /*) | (0/3/0) (0/3/0) (0/3/0) (0/3/0) (0/3/0) (0/3/0)

Key:
(1) rules.ZeroR " 48055541465867954
(2) functions.Logistic '-R 1.0E-8 -M -1' 3932117032546553727
(3) meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W meta.Bagging -- -P 100 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.0010 -N 3 -S 1 -L -1' -7378107808933117974
(4) meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W meta.Dagging -- -F 10 -S 1 -W functions.SMO -- -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -C 250007 -E 1.0\"' -7378107808933117974
(5) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"GeneticSearch -Z 20 -G 20 -C 0.6 -M 0.033 -R 20 -S 1\" -W meta.AdaBoostM1 -- -P 100 -S 1 -I 10 -W meta.Dagging -- -F 10 -S 1 -W functions.SMO -- -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -C 250007 -E 1.0\"' -5951805453487947577
(6) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"GeneticSearch -Z 20 -G 20 -C 0.6 -M 0.033 -R 20 -S 1\" -W meta.AdaBoostM1 -- -P 100 -S 1 -I 10 -W meta.Bagging -- -P 100 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.0010 -N 3 -S 1 -L -1' -5951805453487947577
(7) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"GeneticSearch -Z 20 -G 20 -C 0.6 -M 0.033 -R 20 -S 1\" -W functions.Logistic -- -R 1.0E-8 -M -1' -5951805453487947577

A partir de estos resultados se ha realizado la siguiente tabla con el porcentaje de aciertos para poder ver estos resultados con un mejor detalle.

	% ACIERTOS
	SUJETO1
ZEROR	**40.67**
LOGISTIC	**74.33**
LOGISTIC + SELECCIÓN	**75.53**

3. Test

A continuación cogeremos el mejor resultado en media de los experimentos realizados para realizar el test sobre el conjunto de datos B. Obteniéndose los resultados siguientes:

ALGORITMO	% ACIERTOS EXPERIMENTOS	%ACIERTOS TEST	%ACIERTOS MEDIA EN TEST
LOGISTIC	**74.33 / 62.02 / 60.55**	**73.5731 / 60.6855 / 46.6456**	**60.34%**

Comparando nuestros resultados con los obtenidos en la competición BCI, se puede observar como habríamos obtenido el 11º puesto, con lo cual podemos decir que no está nada mal. Además se puede suponer que este algoritmo es bueno en este problema concreto. Se puede observar también como los peores resultados obtenidos son con el tercer sujeto, aunque si nos damos cuenta en la competición los resultados con este sujeto no eran buenos tampoco para ninguno de los algoritmos propuestos.

ANEXO 2: MANUAL DE USUARIO

Este anexo contiene todo lo relacionado con la compilación y ejecución del software realizado, así como las herramientas necesarias para su correcto funcionamiento.

1. *Generar fichero con productos y clases*

Lo primero que tenemos que tener en cuenta es la generación de un fichero correcto con los atributos originales, así como los productos para que el software funcione correctamente. Para que los ficheros sean entendibles por el software es necesario que los datos del fichero se encuentren separados por espacios o tabuladores y que las clases se encuentren en binario {0,1} para el caso de los dominios con dos clases, y {00, 01, 10} para los dominios con tres clases. También habrá que tener en cuenta que los ficheros no deberán contener cabeceras, ni ningún tipo de texto excepto los datos. Para ello, el código escrito en Java Main.java, es necesario introducir los siguientes parámetros por línea de comandos al llamar al programa:

NumeroAtributos FicheroOrigen FicheroDestino

Donde:

- NumeroAtributos es el número de atributos original que contiene el fichero
- FicheroOrigen es un fichero de datos sin cabeceras y con cada atributo separado por un espacio o tabulador, así como cada dato separado por un fin de línea y con las clases definidas correctamente.
- FicheroDestino es el fichero donde se guardarán los datos que se generen.

Aparte del fichero con los atributos producto también será necesario disponer de otro fichero de datos con los atributos originales y las clases. Después será necesario dividir los conjuntos de datos en entrenamiento, test y validación para ambos ficheros. Si los datos ya venían divididos en estos tres conjuntos, habría que ejecutar el código anterior tres veces para generar los ficheros de entrenamiento, test y validación. En total obtenemos seis ficheros que serán la entrada para nuestro siguiente paso.

2. *Ejecutar perceptron simple*

Para ejecutar el perceptron simple y evaluar su capacidad se debe usar el código que se encuentra la carpeta perceptronesSimples y copiar los ficheros de entrenamiento, test y validación simples en la carpeta de ./input_data. Después se debe abrir el fichero correspondiente a la combinación de perceptrones que queramos usar y modificar dentro del fichero los valores en función del dominio para el que se vaya a ejecutar:

```
#define NUM 8         /* Numero de atributos. */
#define LIMIT 700     /* Numero de patrones del fichero de entrenamiento.
Numero de lineas del fichero de datos*/
#define LIMIT_TEST 300     /* Numero de patrones del fichero de test.  Numero
de lineas del fichero de datos*/
#define LIMIT_VALIDA 1000  /* Numero de patrones del fichero de validacion.
Numero de lineas del fichero de datos*/
#define SESSIONS 300 /* Numero de sesiones de entrenamiento del perceptron */
#define REPETICIONES 10 /* Numero de veces que se repetira el PS. */
```

```
char* test= "./input_data/zerosSimpleTest.txt";
char* seleccion= "./input_data/zerosSimpleTrain.txt";
char* validacion="./input_data/zerosSimpleValidacion.txt";
```

Existen varias implementaciones de perceptrones simples que son las siguientes:

- PerceptronSimple: Se trata de un perceptron simple para dominios con dos clases.
- Perceptron2PS: Se trata de dos perceptrones simples combinados para dominios con tres clases.
- Perceptron3PS: Se trata de tres perceptrones simples combinados para dominios con tres clases.
- PerceptronSigmoidal: Se trata de un perceptron simple con una función de activación sigmoidal para dominios con tres clases.

Después de cambiar estos valores será necesario compilar y ejecutar el código, para ello deberán ejecutarse estas dos instrucciones:

Compilar: gcc –o perceptronSimple perceptronSimple.c
Ejecutar: ./perceptronSimple

Para compilar la implementación realizada sobre el perceptron que utiliza la función sigmoidal a la salida será necesario ejecutar las siguientes instrucciones:

Compilar: gcc –lm –o perceptronSigmoidal perceptronSigmoidal.c
Ejecutar: ./perceptronSigmoidal

Como resultado de la ejecución se obtiene el porcentaje de aciertos para entrenamiento, test y validación.

3. Ejecutar programa principal

El programa principal contiene el algoritmo NSGAII, un algoritmo multiobjetivo que ya se explicó en detalle en el apartado 3.4 “NSGAII”, y cuyo código es de libre distribución y se encuentra en http://www.iitk.ac.in/kangal/codes.shtml . Fue realizado por K. Deb y sus alumnos del Laboratorio de Algoritmos Genéticos del Instituto Tecnológico Kanpur en India [Deb00]. Se deberán copiar los ficheros con los atributos producto a la carpeta ./input_data, después será necesario especificar el tipo de multiobjetivo que se desea realizar, para ello será necesario cambiar en el fichero problemdef.c el problema que se desea ejecutar. Existen dos opciones:

- Minimizar el número de fallos total y el número de atributos usados. Para esta opción el código de problemdef.c deberá estar en bci, como se muestra a continuación.

```
# define bci
/* # define bci2 */
```

- Minimizar el número de fallos para cada una de las clases. Para esta opción el código de problemdef.c deberá estar en bci2 para el dominio de BCI, y se debe escoger la opción de bci para los dominios con dos clases.

A la hora de realizar cualquier prueba será necesario tener en cuenta los siguientes factores que serán propias del algoritmo NSGAII o del clasificador utilizado como son los perceptrones simples:

- Probabilidad de mutación: Se trata de la probabilidad de mutación del algoritmo genético que utiliza el algoritmo NSGAII, después de realizar algunas pruebas se consideró que este valor permaneciera constante en 0.0125.

- Probabilidad de cruce: Se trata de la probabilidad de cruce (crossover) del algoritmo genético que utiliza el algoritmo NSGAII, después de realizar algunas pruebas se consideró que este valor permaneciera constante en 0.9.

- Generaciones: Número de veces que se ejecutara el algoritmo NSGAII, este valor se deberá modificar para realizar las distintas pruebas.

- Individuos: Número de individuos que tendrá el algoritmo NSGAII, y debido a como se ha implementado este algoritmo este número debe ser múltiplo de 4.

- Iteraciones: se trata del número de ciclos de aprendizaje que realizarán los dos perceptrones simples combinados, el valor por defecto será de 300.

- Semilla de inicialización: Variable del algoritmo NSGAII que se fija a 0,5 y se mantendrá este valor para todas las ejecuciones.

- Razón de aprendizaje: Solo para el caso de la combinación de tres perceptrones simples.

Se debe abrir el fichero perceptronBest.c y modificar los valores en función del dominio para el que se vaya a ejecutar, y cambiar los valores de las iteraciones y la razón de aprendizaje:

```
#define NUM 36         /* Numero de atributos. */
#define LIMIT 700     /* Numero de patrones del fichero de entrenamiento.
Numero de lineas del fichero de datos*/
#define LIMIT_TEST 300     /* Numero de patrones del fichero de test.  Numero
de lineas del fichero de datos*/
#define LIMIT_VALIDA 1000  /* Numero de patrones del fichero de validacion.
Numero de lineas del fichero de datos*/
#define SESSIONS 300 /* Numero de sesiones de entrenamiento del perceptron */
#define REPETICIONES 10 /* Numero de veces que se repetira el PS. */

char* test= "./input_data/zerosSimpleTest.txt";
char* seleccion= "./input_data/zerosSimpleTrain.txt";
char* validacion="./input_data/zerosSimpleValidacion.txt";
```

También será necesario abrir el fichero bci.in dentro de la carpeta ./input_data y modificar los valores de la población, número de generaciones, probabilidad de cruce, probabilidad de mutación, atributos y objetivos, dejando igual el resto de valores.

```
24 → Poblacion(múltiplo de 4)
100 -> generaciones
2 -> número de objetivos
0 -> valor inicial del primer objetivo
```

```
0 -> valor inicial del segundo objetivo
1
36 0 1 –> el primer valor es el número de atributos
0.9 -> probabilidad de cruce
0.0125 -> probabilidad de mutación
```

Para ejecutar el código simplemente bastará con realizar las siguientes instrucciones

Compilar: make clean
Make all
Ejecutar: ./nsga2r 0.5 < ./input_data/bci.in

Existen algunas ocasiones en que la ejecución puede durar mucho tiempo, si no se quiere mantener la ventana del putty abierta (se puede apagar el ordenador) mientras se realiza la ejecución se puede utilizar la instrucción:

Nohup ./nsga2r 0.5 < ./input_data/bci.in &

Como salida de la ejecución de este programa se obtienen los siguientes ficheros de salida:

- Params.out: Contiene los parámetros con los que se ha ejecutado el programa, tales como el tamaño de la población o el número de generaciones del algoritmo genético multiobjetivo.
- Inicial_pop.out: Contiene los individuos y su fitness de la población inicial.
- Final_pop.out: Contiene los individuos y su fitness de la última generación del programa.
- All_pop.out: Contiene los individuos y su fitness de todas las generaciones del programa.
- Best_pop.out: Contiene los individuos que se encuentran en el frente de la última generación del programa, así como el fitness para los conjuntos de entrenamiento, test y validación.

Por pantalla aparecerán los datos del número total de ejemplos para los conjuntos de entrenamiento, test y validación, que será necesario tener en cuenta para posteriores fases.

Existe una opción para permitir que la ejecución comience con determinados individuos. Para ello, fue necesario modificar la implementación del código de NSGAII de manera que permitiera en un fichero denominado "./input_data/inicializa_pob.txt" introducir cierto número de individuos como población inicial. Este código se ha realizado de manera que se pueden inicializar ciertos individuos a través de fichero y no tiene porque ser toda la población. De esta forma por ejemplo para una población inicial de 100 individuos se podrían inicializar solamente 50 o los que se desearan. La inicialización por fichero es opcional y basta con que el fichero se borre para que el código del NSGAII inicialice todos los individuos aleatoriamente. Con esta opción además podríamos continuar la ejecución de una prueba anterior. Esto podría ser útil si nos damos cuenta de que la prueba anterior necesitaba más generaciones o si por cualquier problema en el hardware cualquier prueba no termina y se queda a media ejecución.

4. Obtener frentes de los ficheros

Se utilizan para esta fase los ficheros all_pop.out y best_pop.out de la ejecución del código anterior. Se obtendrán tanto la evolución del frente a lo largo de las generaciones para el conjunto de entrenamiento, como el frente final para los conjuntos de entrenamiento, test y validación. Además de comparaciones de frentes dos a dos para los casos de tres objetivos.

Para obtener la evolución de los frentes será necesario ejecutar el código Java PintaFrentes.java introduciendo los siguientes parámetros de forma correcta.

ficheroOrigen generaciones atributos objetivos objetivo1 objetivo2 [objetivo3]

Donde:
ficheroOrigen será all_pop.out
generaciones es la frecuencia de generaciones con la que se pintarán los frentes, tiene que ser distinta de 1.
Atributos es el número de atributos total, es decir, sumando los atributos producto que contiene el fichero.
Objetivos es el número de objetivos que se desea minimizar
Objetivo1 es el primer objetivo del programa, ya sea el número de atributos o el número total de ejemplos para la clase1.
Objetivo2 es el segundo objetivo del programa, ya sea el número total de ejemplos del conjunto de entrenamiento o el número total de ejemplos para la clase2.
Objetivo3 es opcional y sólo será necesario para casos de 3 objetivos, siendo este el número de ejemplos total para la clase 3.

Como salida se obtendrá una carpeta llamada frenteall donde estarán todos los ficheros de los frentes.

Para obtener los ficheros de los frentes para los conjuntos de entrenamiento, test y validación del frente de la última generación, será necesario ejecutar un fichero distinto para cada caso:

- En el caso de que se quiera obtener el frente para el primer caso, es decir, el objetivo de minimizar el número de atributos y el porcentaje de fallos total; sobre los dominios de dos o tres clases, o se ejecute un dominio de tres clases sobre el segundo caso, es decir minimizar el porcentaje de acierto para cada clase, se deberá ejecutar el código Java Frentes2.java introduciéndose como parámetro:

fichOrigen objetivos atributos totalTrain totalTest totalValida [clase1 clase2 clase3 claseTest1 claseTest2 claseTest3 claseValida1 claseValida2 claseValida3]

Donde:
ficheroOrigen será best_pop.out
Objetivos es el número de objetivos que se desea minimizar
Atributos es el número de atributos que contiene el fichero.
totalTrain es el número de ejemplos total del fichero de entrenamiento
totalTest es el número de ejemplos total del fichero de test

totalValida es el número de ejemplos total del fichero de validación

Los siguientes atributos son opcionales, ya que sólo son necesarios en el caso de dominios con tres clases para el segundo caso.

Clase1 número total de ejemplos de la clase1 en el fichero de entrenamiento.
Clase2 número total de ejemplos de la clase2 en el fichero de entrenamiento.
Clase3 número total de ejemplos de la clase3 en el fichero de entrenamiento.
ClaseTest1 número total de ejemplos de la clase1 en el fichero de test.
ClaseTest2 número total de ejemplos de la clase2 en el fichero de test.
ClaseTest3 número total de ejemplos de la clase3 en el fichero de test.
ClaseValida1 número total de ejemplos de la clase1 en el fichero de validación.
ClaseValida2 número total de ejemplos de la clase2 en el fichero de validación.
ClaseValida3 número total de ejemplos de la clase3 en el fichero de validación

Como salida se obtendrán los ficheros frenteTrain.txt, frenteTest.txt y frenteValida.txt que representan los puntos del frente de la última generación en porcentaje.

- En el caso de que se quiera obtener el frente para el segundo objetivo, es decir, el objetivo de minimizar el porcentaje de fallos para cada una de las clases y el dominio sea distinto del BCI se deberá ejecutar el código Java Frentes3.java con los siguientes parámetros:

fichOrigen atributos totalTrain totalTest totalValida clase1 clase2 claseTest1 claseTest2 claseValida1 claseValida2

Donde:
ficheroOrigen será best_pop.out
Atributos es el número de atributos que contiene el fichero.
totalTrain es el número de ejemplos total del fichero de entrenamiento
totalTest es el número de ejemplos total del fichero de test
totalValida es el número de ejemplos total del fichero de validación
Clase1 número total de ejemplos de la clase1 en el fichero de entrenamiento.
Clase2 número total de ejemplos de la clase2 en el fichero de entrenamiento.
ClaseTest1 número total de ejemplos de la clase1 en el fichero de test.
ClaseTest2 número total de ejemplos de la clase2 en el fichero de test.
ClaseValida1 número total de ejemplos de la clase1 en el fichero de validación.
ClaseValida2 número total de ejemplos de la clase2 en el fichero de validación.

Como salida se obtendrán los ficheros frenteTrain.txt, frenteTest.txt, frenteValida.txt, que representan los puntos del frente de la última generación en porcentaje. Además aparece también el frenteTotal.txt en el que aparecen el número de fallos total para los conjuntos de entrenamiento, test y validación junto con el número de individuo del fichero "best_pop.out" y que aparecen ordenados por su porcentaje de aciertos en validación.

- Para el segundo caso con dominios de tres clases puede ser interesante ver el comportamiento de los objetivos comparándolos dos a dos, es decir, aparecen

combinados el objetivo 1y2, 1y3, 2y3, de manera que nos podamos hacer una mejor idea a la hora de pintar los frentes de cómo se distribuyen estos objetivos y cuál puede ser el objetivo que mejor o peor resultado obtiene. Para ello será necesario ejecutar el código Java Frentes.java con los siguientes parámetros:

fichOrigen objetivos clase1 clase2 clase3 claseTest1 claseTest2 claseTest3 claseValida1 claseValida2 claseValida3

Donde:
ficheroOrigen será best_pop.out
Objetivos es el número de objetivos que se desea minimizar
Clase1 número total de ejemplos de la clase1 en el fichero de entrenamiento.
Clase2 número total de ejemplos de la clase2 en el fichero de entrenamiento.
Clase3 número total de ejemplos de la clase3 en el fichero de entrenamiento.
ClaseTest1 número total de ejemplos de la clase1 en el fichero de test.
ClaseTest2 número total de ejemplos de la clase2 en el fichero de test.
ClaseTest3 número total de ejemplos de la clase3 en el fichero de test.
ClaseValida1 número total de ejemplos de la clase1 en el fichero de validación.
ClaseValida2 número total de ejemplos de la clase2 en el fichero de validación.
ClaseValida3 número total de ejemplos de la clase3 en el fichero de validación.

Como salida se obtendrán los ficheros frenteTrain12.txt, frenteTrain13.txt, frenteTrain23.txt, frenteTest12.txt, frenteTest13.txt, frenteTest23.txt, frenteValida12.txt, frenteValida13.txt y frenteValida23.txt que representan los puntos del frente en porcentaje para cada fichero y cada uno de los objetivos, por ejemplo el fichero frenteTrain12.txt contendrá los puntos de los objetivos 1 y 2 para el fichero de entrenamiento.

5. *Pintar los frentes*

Se utiliza como entrada los frentes la carpeta frenteall donde se encuentran los ficheros que representan la evolución de los frentes y los ficheros con los frentes de la última generación. Para poder obtener los frentes pintados será necesario copiar los ficheros que se encuentran en la carpeta frenteall donde se encuentre el fichero plot.sh y después en Linux ejecutar el comando ./plot.sh, será necesario tener instalado el gnuplot para que la ejecución funcione correctamente. De la misma manera, se borran los ficheros que estaban dentro de la carpeta frenteall y se copian los frentes finales para los conjuntos de entrenamiento, test y validación, y se vuelve a ejecutar el comando ./plot.sh. Como salida de ambas ejecuciones se obtendrá un fichero llamado fronts.ps en el que aparecen los frentes pintados. En caso de que se desee obtener un frente que no se encuentre en la escala 0:100, sino que el mismo programa genere la escala adecuándose a los datos, será necesario abrir el fichero plot.sh y en la línea “echo $command | gnuplot fronts.plt –“cambiar por la siguiente línea "echo $command | gnuplot fronts2.plt –“. Para el caso de las pruebas que tengan tres objetivos y se quieran representar los tres objetivos, será necesario abrir el mismo fichero y cambiar la línea “command="plot "” por “command="splot "”.

6. *Herramientas utilizadas*

A continuación se procede a comentar brevemente cada una de las herramientas que han sido utilizadas en la realización de este proyecto, así como una breve descripción sobre cada una de ellas:

- Weka: (Waikato Environment for Knowledge Analysis - Entorno para Análisis del Conocimiento de la Universidad de Waikato) es un conocido software para aprendizaje automático y minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Es un programa de libre distribución y se puede descargar de http://www.cs.waikato.ac.nz/ml/weka/.

- Java 1.6: Lenguaje de programación regido por el paradigma de la orientación a objetos, y que es de libre distribución, y multiplataforma. Su software se puede descargar gratuitamente de la página web http://developers.sun.com/downloads/.

- Netbeans 5.5.1: Se trata de una plataforma para el desarrollo de aplicaciones de usando Java y un entorno de desarrollo integrado (IDE). Esta plataforma permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. NetBeans es un proyecto de código abierto y cuyo software se puede descargar de la página web http://www.netbeans.org/.

- C: Es un lenguaje de programación orientado a la implementación de Sistemas Operativos, concretamente Unix. C tiene la característica de ser eficiente en el código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

- Gnuplot: Se trata de un programa muy flexible para generar gráficas de funciones y datos. Este programa es compatible con la mayoría de sistemas operativos. Puede devolver sus resultados directamente en pantalla, o en multitud de formatos y se puede usar interactivamente o en modo por lotes. Es un programa de libre distribución y código abierto y se puede descargar de http://www.gnuplot.info/download.html

- Editplus: Se trata de un editor de texto, con un cierto parecido al bloc de notas de Windows pero mucho más potente y que permite visualizar con gran facilidad numerosos ficheros de texto en cualquier formato. Este programa necesita una licencia para utilizarse, pero se puede descargar una demo de treinta días en la dirección http://www.editplus.com/download.html.

- Software NSGAII: se trata de un algoritmo genético multiobjetivo que ya se explicó en detalle en el apartado 3.4 “NSGAII”, y cuyo código que es de libre distribución y se encuentra en http://www.iitk.ac.in/kangal/codes.shtml fue realizada por K. Deb y sus alumnos del Laboratorio de Algoritmos Genéticos del Instituto Tecnológico Kanpur en India.

- Winscp: Se trata de una aplicación que actúa como un cliente SFTP gráfico para Windows que emplea SSH. El anterior protocolo SCP también puede ser empleado. Su función principal es facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios SSH. Se trata de una aplicación de software libre y se puede descargar de http://winscp.net/eng/download.php.

- Beyond Compare: Programa muy útil para la comparación de ficheros y directorios. Se necesita licencia para su utilización pero es posible descargar versiones de prueba gratuitas.

- Putty: Se trata de una aplicación que actúa como un cliente SSH, Telnet, rlogin y TCP raw que se encuentra disponible para varias plataformas. Se trata de un programa de libre distribución que se puede conseguir en la dirección http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

- Microsoft Office 2003: Es un conjunto de programas de ofimática, compuesta básicamente por aplicaciones de procesamiento de textos, plantilla de cálculo y programa para presentaciones. Fue desarrollada por la empresa Microsoft y es necesario disponer de una licencia para poder utilizar este software.

ANEXO 3: GESTIÓN DEL PROYECTO

En el presente anexo se describe la metodología utilizada en el proyecto además, se especifican tanto los recursos humanos y materiales como la tecnología necesaria para su éxito, y para finalizar, se da un presupuesto que refleja el coste total que supondrá el desarrollo del proyecto.

1. *Método de trabajo*

A la hora de realizar un proyecto de investigación varía la forma de trabajo de por ejemplo la realización de un proyecto software, ya que el tiempo necesario para realizar cada tarea, así como los problemas que pueden surgir no son nada fáciles de determinar.

Para ello, se debe ver de forma abstracta cual son las tareas a desarrollar y realizar una división lógica de las mismas y cual serán los criterios de transición entre las mismas. Debido al desconocimiento inicial de las ampliaciones necesarias que serían necesarias para conseguir realizar el proyecto se decidió utilizar un ciclo de vida evolutivo como en el que se presenta en la figura 101 y en la que se puede comprobar cómo es necesario finalizar cada una de las fases para continuar con la siguiente, ya que los resultados de una etapa son los que nos determinan la anterior.

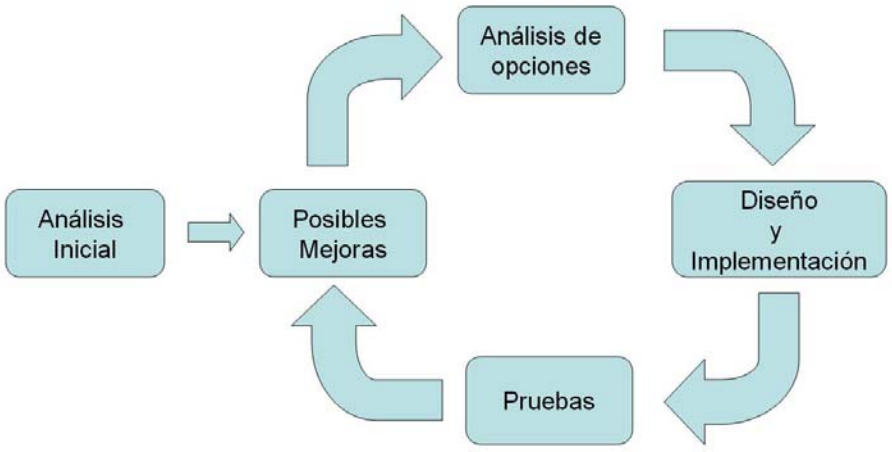


Ilustración 101. Ciclo de vida

2. *Presupuesto*

Para realizar el cálculo del presupuesto es necesario tener en cuenta diversas consideraciones:

- Jornada laboral de cinco horas.
- Cinco días laborables a la semana (excluidos fines de semana y festivos).
- El periodo de realización será:

  - Fecha de inicio: 3 de Marzo de 2008.
  - Fecha de fin: 16 de Febrero de 2009.
- En los meses de junio a octubre no se realizó prácticamente ninguna actividad sobre el presente proyecto y por tanto, no se imputará ninguna hora con cargo al proyecto.
- Los días festivos considerados son:
  - Semana Santa: 19, 20 y 21 de Marzo.
  - Puente de Mayo: 1 y 2 de Mayo.
  - Navidades: 25 de Diciembre, 1 y 6 de Enero.
- Es necesario un ingeniero informático especializado en inteligencia artificial para realizar el proyecto.
- Coste de personal por hora de trabajo:
  - Ingeniero informático: 45€/hora.

Por lo tanto, el cómputo total de horas dedicadas al proyecto es 660 horas, por lo que el proyecto ha tenido una duración total de 132 días reales.

Costes de hardware para el desarrollo del proyecto.

- Intel Pentium a 3,01 GHz, 1,00 GB de RAM.
- Portátil Intel Dual Core con procesadores a 1.6 GHz, 1,00 GB de RAM.
- Máquina Linux 2 del departamento de IA para realizar las pruebas.

Los costes de las dos primeras máquinas amortizadas a tres años tendrían un coste de 300€, así como la tercera máquina se podría decir que ha sido utilizada en régimen de alquiler y compartido con otras personas para la realizaciones de las pruebas del proyecto y que es difícil calcular su coste, aunque se podría estimar en unos 200€/mes, teniendo en cuenta que el primer mes no sería necesario.

Hardware	Coste	Coste Total
Intel Pentium a 3,01 GHz, 1,00 GB de RAM	125€ amortizado a tres años	125€
Portátil Intel Dual Core con procesadores a 1.6 GHz, 1,00 GB de RAM	175€ amortizado a tres años	175€
Linux 2	200€/mes	1000€
	Coste total	**1300€**

Tabla 18. Coste hardware

A continuación aparece el coste total del proyecto, teniendo en cuenta el coste de los recursos humanos y materiales necesarios para la realización del mismo, así como los riesgos y los beneficios asociados al mismo. Se va considerar el riesgo máximo del 15%, mientras que el beneficio será del 5%.

CONCEPTO	COSTE (€)
Recursos humanos	29.700
Hardware	1.300
Riesgo (15%)	4.650
Beneficios (5%)	1.782,5
Coste total	**37.432,5**
I.V.A. (16%)	5.989,2
Coste total con I.V.A.	**43.421,7**

Tabla 19. Coste total del proyecto

Por tanto el coste total del proyecto sería de **43.421,70€,** cuarenta y tres mil cuatrocientos veintiún euros con setenta céntimos de euro.